ST. LOUIS SEARS

PRESENTS

THE BEST OF

THE COMPUTER BRIDGE

INDEX

MEMBER WRITTEN SOFTWARE NOT APPEARING IN PRINT

BIRTH/WRAP - PERSONALIZED WRAPPING PAPER
CABIN WRAP - " " " " " "
CHILL - WINDCHILL CALCULATOR
EGG HUNT - TEXT ADVENTURE BY GEORGE PASCHETTO
STL/MAIL - OUR MAIL PROGRAM

PROGRAMS

BASEMENT INVADERS

by George Pascetto

Editors note: This program was give out by George as part of his Introduction to Extended BASIC course. If you'd like to learn more contact George at 352-3409 about his next scheduled course offerings. Watch those roaches!!

```
100 DATA In BASEMENT INVADERS you,must wipe out the roaches,that are attacking you.,,You move your nozzle
with,the S and D keys and fire
110 DATA with Q.,,CAUTION! If you fire five,times without a hit--you,lose pressure and must,start again.
120 CALL CLEAR :: FOR X=3 TO 14 :: READ B$ :: DISPLAY AT(X,3):B$ :: NEXT X
130 DISPLAY AT(22,5):"PRESS ENTER)>" :: ACCEPT AT(22,18):B$
140 CALL SCREEN(15):: OPTION BASE 1 :: DIM B(9),D(10)
150 CALL CHAR(136,"180C0C1830606030", 128,"FFFFFFFF7E3C1818", 34, "43242211BD826819" :: CALL COLOR(14,9,1)
160 A$(1)="AIQBJRAIQBJRAIQBJRAIQBJRAIQ" :: CALL CLEAR :: E=2
170 CALL COLOR(5,1,1,6,1,1,7,1,1):: C$="817E18FF3CFF187E24997E18FF187E99" :: CAL L CHAR(65,C$,73,C$,81,C$)
180 C=16 :: F=5 :: M=2 :: CALL SPRITE(#1,128,16,2,121)
190 FOR G=1 TO E :: FOR V=-1 TO 1 STEP 2 :: PRINT A$(M+V):: CALL SOUND(5,-2,0):: R=R+1 :: IF R-H=23 THEN 350
ELSE GOSUB 220
200 FOR I=6-V TO 6 STEP V :: CALL COLOR(I,1,1,I+V,13,1):: CALL SOUND(5,-2+V,0):: F=I+V-5 :: GOSUB 220
210 NEXT I :: NEXT V :: NEXT G :: M=3 :: GOTO 190
220 FOR T=0 TO 2 :: CALL KEY(1,K,S)
230 IF K=18 THEN 240 ELSE J=(K=2)-(K=3):: GOTO 280
240 N=INT(C/3):: IF B(N)=E THEN 300 ELSE L=24+B(N)*2-R :: IF L>24 THEN L=24
250 CALL VCHAR(2,C,136,L-2):: CALL SOUND(10,880,0):: CALL VCHAR(2,C,32,L-2):: M= M+1 :: IF M=6 THEN 350
260 IF (C<>N*3+F)+(INT(R/2)=B(N))THEN 300 ELSE CALL HCHAR(L,N*3,32,3):: B(N)=B(N )+1 :: D(B(N))=D(B(N))+1 ::
CALL SOUND(10,990,0)
270 M=0 :: IF D(B(N))<9 THEN 300 ELSE H=H+2 :: IF H=E*2 THEN 310 ELSE 300
280 IF J=0 THEN 300 ELSE C=C+J :: IF C=2 THEN C=29 ELSE IF C=30 THEN C=3
290 CALL LOCATE(#1,2,C*8-7):: CALL SOUND(10,220,0)
300 NEXT T :: RETURN
310 DISPLAY AT(20,5):"YOU WIN LEVEL ";E/2;"!" :: E=E+2 :: IF E<10 THEN 370 ELSE DISPLAY AT(22,2):"AGAIN PRESS 1,
END=2."
320 CALL KEY(0,K,S):: IF (K<49)+(K>50)THEN 320 ON K-48 GOTO 390,330
330 PRINT : : :" BYE BYE. ": : :
340 END
350 CALL CLEAR
360 PRINT : : : : : : : :"YOU LOSE.": : : : : :
370 PRINT " PRESS ANY KEY."
380 M=0 :: CALL KEY(0,K,S):: IF S=0 THEN 380 ELSE CALL CLEAR :: CALL SOUND(10,-1 ,0)
390 FOR X=1 TO 9 :: B(X)=0 :: NEXT X :: FOR X=1 TO 10 :: D(X)=0 :: NEXT X
400 L,H,R,N,F=0 :: CALL COLOR(5,1,1,6,1,1,7,1,1,8,1,1):: GOTO 180
```

VIDEOTAPE FILES

By Max Hyde

For those who have VCR's and a library of tapes, some method of cataloging is needed. Here is a program that will satisfy many of your needs, especially if you have printing facilities.

A discussion of the program may save you problems later, even though the instructions are in the program. First, write, of "load" the listed program into your console. Notice first, line 180. Since I use a serial connected printer, I've written the program for this option. If you "LOAD" the program from cassette or disk, EDIT this line (if needed) to match your option - "PIO", or whatever.

Line 190 - you can list your "Titles" anyway you like, not necessarily as shown in my example, you may want to show "Control Index" as well. You are limited to the usual 4 lines per entry. The reason I've specified "3 spaces" in the example (you can use any number as a separator) is because to "search", "change", or "delete", a title - IT MUST BE TYPED EXACTLY as listed, or it will not show up.

Line 220 explains use of line 480. As written, it puts 4 titles on the monitor each time. Change line 480 as desired. Since all of my titles are "one liners", I use 4 titles on the screen at a time. If you opt to make this greater, the upper titles will scrool off the screen.

Line 280 - This limits the number of titles on any one given listing. I have shown 300, but frankly I don't know if this is true. Naturally shorter titles will extend this but it will vary with system configuration.

Now LOAD the program and RUN. An option screen will allow you to select instructions or not. Press C to continue and the main option screen will appear.

(EDITORS NOTE: A complete list of instructions is available on request. The instructions were just to lengthy to print in the newsletter since I try to maintain a 12 page max for postage reasons.)

```
100 CALL CLEAR
110 PRINT " VIDEODISC
FILE ":: :
120 PRINT " BY MAX HY
DE ":: :: :: :: :: :: :
130 PRINT "NEED INSTRUCTIONS
? (Y/N)"
140 CALL KEY(0,K,S)
150 IF S=0 THEN 140
160 IF K<>89 THEN 280
170 CALL CLEAR
180 PRINT "PRINT OPTION IS S
ET AS ""SERIAL"" RS232.BA=60
0. CHANGE LINE 300, IF YO
UR DEVICE IS DIFFERENT.":
190 PRINT "TO ENTER YOUR TAP
ES, LIST TITLE , 3 SPACES
, DISC#, 3 SPACES,THEN TIME
(OR INDEX)":
200 PRINT " KEEP # OF SPACES
UNIFORM. WHEN SEARCHING OR
DELETING, TYPE TITLE EXACT
, INCLUDING SPACES":
210 PRINT "EXAMPLE ";"KOJA
K D#1 1:30":
220 PRINT "TO CHANGE LENGTH
OF VIEWED TITLES AT ONE TIM
E,CHANGE LINE 480":
230 PRINT "PRESS C TO CONTIN
UE"
240 CALL KEY(0,K,S)
250 IF K=67 THEN 280
260 IF K<>67 THEN 240
270 IF S=0 THEN 240
280 DIM T$(300)
290 CALL CLEAR
300 O$="RS232.BA=600"
310 W$=" PLEASE WAIT..
. THE PRINTER IS WORK
ING"
320 CALL CLEAR
330 PRINT " MAIN INDE
X":: :
340 PRINT "PRESS TO":: :
350 PRINT " 1 = VIEW TITL
ES":" 2 = SEARCH FOR A TI
TLE":" 3 = ADD TITLES":"
4 = CHANGE TITLES"
360 PRINT " 5 = DELETE TI
TLES":" 6 = ALPHABETIZE L
IST":" 7 = SAVE TITLE FIL
E":" 8 = LOAD TITLE FILE"
370 PRINT " 9 = PRINT TIT
LES":" 10 = FINISH SESSION
":: :
380 INPUT V
390 IF V>10 THEN 380
400 IF V<1 THEN 380
410 CALL CLEAR
420 ON V GOSUB 440,570,740,9
50,1260,1420,1680,1760,1910,
2040
430 GOTO 320
440 T=0
450 FOR I=1 TO N
460 T=T+1
470 PRINT T$(I):: :
480 IF T<4 THEN 540
490 PRINT ::
500 PRINT " $PRESS ENTER TO
CONTINUE$";" $""M"",ENTER FO
R MAIN INDEX$":
510 INPUT A$
520 IF A$="M" THEN 560
530 T=0
540 NEXT I
550 INPUT " $END OF FI
LE$ $PRESS ENTER TO
CONTINUE$":X$
560 RETURN
570 INPUT "TITLE? ":N$
580 FOR I=1 TO N
590 IF T$(I)<>N$ THEN 700
600 PRINT :: :" IS THE TIT
LE:":" ";T$(I):
610 INPUT " (Y/N)?":X$
620 IF X$="N" THEN 700
630 PRINT :: :T$(I):: :
640 INPUT " DO YOU WISH TO
PRINT A LIST? (Y/N)":L$
650 IF L$<>"Y" THEN 670
660 GOSUB 2000
670 INPUT "SEARCH MORE TITLE
S? (Y/N)":S$
680 IF S$="Y" THEN 570
690 GOTO 730
```

3

```
700 NEXT I
710 PRINT : : :" THE ";N$:"
YOU ARE SEARCHING FOR":"
IS NOT IN THIS FILE.": : :
720 GOTO 670
730 RETURN
740 A=N+1
750 FOR I=A TO 300
760 CALL CLEAR
770 PRINT : : : :"ENTER DATA
; ";"#";I;" (MAX:300)": : :
780 PRINT " TITLE :"
790 INPUT T$(I)
800 V=I
810 CALL CLEAR
820 PRINT "ENTRY";"#";V: : :
830 PRINT "YOU ENTERED:": :"
";T$(V)
840 INPUT "CHANGE ANYTHING?
(Y/N)":C$
850 IF C$<>"Y" THEN 890
860 C=N+1
870 CALL CLEAR
880 GOSUB 1030
890 INPUT "ADD MORE TITLES?
(Y/N)":A$
900 N=N+1
910 IF A$="N" THEN 940
920 NEXT I
930 INPUT " $DATA FILE IS
FULL$ $PRESS ENTER TO
CONTINUE$":E$
940 RETURN
950 PRINT " TITLE TO BE CHA
NGED:": : : :
960 INPUT C$
970 CALL CLEAR
980 FOR C=1 TO N+1
990 IF T$(C)=C$ THEN 1000 EL
SE 1210
1000 PRINT "IS THE TITLE:":"
";T$(C): :
1010 INPUT " (Y/N)?":E$
1020 IF E$="Y" THEN 1030 ELS
E 1210
1030 PRINT : : : : : : :"PRE
SS TO CHANGE": :
1040 PRINT " 1 = TITLE":
:" 2 = NO CHANGE": :
1050 R=C
1060 N$=" $ENTER THE NEW DA
TA:"
 1070 INPUT P
 1080 CALL CLEAR
 1090 IF P<1 THEN 1070
 1100 IF P>2 THEN 1070
 1110 IF P=2 THEN 1160
 1120 ON P GOSUB 1230
 1130 PRINT : :"MORE CHANGES
FOR:";" ";T$(R): :
1140 INPUT " (Y/N)?":Y$
1150 IF Y$<>"N" THEN 1030
1160 PRINT : : :"CHANGE DATA
FOR OTHER TITLES?": : :
1170 INPUT " (Y/N)":C$
1180 CALL CLEAR
1190 IF C$<>"N" THEN 950
1200 RETURN
1210 NEXT C
1220 RETURN
1230 PRINT "TITLE WAS:": :T$
(R): : :N$
1240 INPUT T$(R)
1250 RETURN
1260 INPUT "TITLE? ":N$
1270 FOR I=1 TO N
1280 IF T$(I)<>N$ THEN 1380
1290 PRINT : : :"IS THE TITL
E:":" ";T$(I): :
1300 INPUT " (Y/N)?":Y$
1310 IF Y$<>"Y" THEN 1380
1320 A=I
1330 FOR D=A TO N
1340 T$(D)=T$(D+1)
1350 NEXT D
1360 N=N-1
1370 GOTO 1390
1380 NEXT I
1390 INPUT "MORE DELETIONS?
(Y/N)":D$
1400 IF D$="Y" THEN 1260
1410 RETURN
1420 PRINT " PLEASE WA
IT...": : :" THE LIST IS BEI
NG ARRANGED": : : : : : : :
: :
1430 B=1
1440 B=2*B
1450 IF B<=N THEN 1440
1460 B=INT(B/2)
1470 IF B=0 THEN 1600
1480 FOR Y=1 TO N-B
1490 X=Y
1500 I=X+B
1510 IF T$(X)=T$(I)THEN 1570
1520 IF T$(X)<T$(I)THEN 1580
1530 GOSUB 1640
1540 X=X-B
1550 IF X>0 THEN 1500
1560 GOTO 1580
1570 GOSUB 1610
1580 NEXT Y
1590 GOTO 1460
1600 RETURN
1610 IF T$(X)<T$(I)THEN 1630
1620 GOSUB 1640
1630 RETURN
1640 N$=T$(X)
1650 T$(X)=T$(I)
1660 T$(I)=N$
1670 RETURN
1680 GOSUB 1870
1690 OPEN #1:F$,INTERNAL,OUT
PUT,FIXED 150
1700 PRINT #1:N
1710 FOR I=1 TO N
1720 PRINT #1:T$(I)
1730 NEXT I
1740 CLOSE #1
1750 RETURN
1760 GOSUB 1870
1770 OPEN #1:F$,INTERNAL,INP
UT ,FIXED 150
1780 INPUT #1:N
1790 FOR I=1 TO N
1800 INPUT #1:T$(I)
1810 NEXT I
1820 CLOSE #1
1830 CALL CLEAR
1840 PRINT " ";F$: :" THIS
FILE HAS";N;"ENTRIES.": :"
$300 ENTRIES IS MAXIMUM$":
: : : : : : : : :
1850 INPUT " $PRESS ENTER TO
CONTINUE$":X$
1860 RETURN
1870 PRINT " WHAT IS THE
NAME OF":" YOUR STORAGE
DEVICE?": :
1880 PRINT " DON'T FORGE
T DSK1.":" IN THE FILE N
AME": : : : : : : : : : : :
1890 INPUT F$
1900 RETURN
1910 PRINT "PRESS ""P"" TO P
RINT": : :
1920 CALL KEY(0,K,S)
1930 IF K<>80 THEN 1920
1940 IF S=0 THEN 1920
1950 PRINT : : : : : : : : :
: : :W$: : : : : : : : : :
1960 FOR I=1 TO N
1970 GOSUB 2000
1980 NEXT I
1990 RETURN
2000 OPEN #2:O$
2010 PRINT #2:TAB(5);T$(I)
2020 CLOSE #2

2030 RETURN
2040 INPUT " DO YOU WI
SH TO TERMINATE THI
S SESSION? (Y/N)":X$2050
CALL CLEAR
2060 IF X$<>"Y" THEN 320
2070 STOP
```

```
% % % % % % % %
% EASY LOADER %
% by CAVE MAN %
% % % % % % % %
```

Where oh where did the CAVE MAN go? And what ever happened to the dbasic? Here I am, but it's gonna take me a bit of a while to document beyond the linked list. In the meantime I thought we might enjoy a little X-BASIC. We will use the most deadly combination of statements known as PEEK and POKE. We will also see how both BASICs store the actual code we enter.

This episode became necessary cause TI left us with a bit of a mystery. In X-BASIC, we can use RUN "DSK1.NAME", but (while A$="DSK1.NAME") you cant use RUN A$. Every once in awhile i come across a program that has overcome the problem, The REMarks give no clue how they do it, but it always uses RUN "DSK1.0123456789" as the last statement in the program. It also uses CALL LOAD statements to place something into memory. Armed with this information , i assumed it was loading the program name into the "0123456789". But where was that string, and how do we find it? The answer is simple, because the system keeps track of where it is with a pointer. To explain this pointer , we have to look at the way the system stores our programs in memory.

Both BASIC's store code in three independent pieces. They are, the Line Number Table, the Numeric Value Table, and the Program Code. Given the chance, the X-BASIC system will store most of this in the expanded 32k of memory, so this program assumes the 32k is present, and will probably not work without it. When the program is running, it uses the Line Number Table to see which line to execute next. The table lists the line numbers, with a pointer to the address of the code for that line. When it is done executing that code, it returns to the table, to look up the next chunk of code. All variables, and arrays are stored in the Numeric Value Table.

The values in the Numeric value Table are pointed to by entries in the Symbol Table, which is located in VDP ram and pointed to by >833E (-31863). This area contains numbers in RADIX 100 notation. Luckily, we dont have to use this area.

The address of the Line Number Table is pointed to by >8330(-31952), with the table end pointed to by >8332 (-31950). While the program is running, address >832E (-31954) points to the line number currently being referenced. The format of the Line Number Table has 2 bytes to represent the line number, followed by 2 bytes for the address of the code for that line number. We will be PEEKing in this area for directions.

Finally we have the Program Space. It starts at the location pointed to by >8332 (-31950). The format of the code in this area is, one byte for the length of the line, followed by the bytes of code, followed by the last byte as >00. We're gonna POKE stuff into this area with CALL LOAD.

In TI BASICs, reserved words are stored as one byte tokens with a decimal value greater than 127. A complete list of these tokens can be found in the COMPUTER BRIDGE (July/Aug 1985, pg 17). These tokens are used by the system as commands, with the rest of the statement occupied with data. Data takes several forms. For example a quoted string is stored as, 200(the token for open quote) , one byte of string length, the string characters (with no closing quote). The next byte will be more data, or another token, or a zero(0), which indicates it is the end of the statement.

Now we can take a look at how our program is gonna use these areas. The first step is to catalog the disk, and list the BASIC type programs. The files with a file type of 5 are stored in an array, B$. Then we PEEK into -31950 to find the start of the code of the last line. The code is stored in the order it is entered, so the last line entered will be the one pointed to by -31950.

I tried substituting just the name of the program into the string of "DSK1.0123456789", but wound up with syntax errors. It seemed that the surest policy was to rebuild the entire line. Line 70 is rebuilt by the program using the following characters. A length byte of 9 plus the length of the program name string. The token for RUN, which is 169. The token for a quoted string, which is 199. The length of the program name + 5 (for DSK1.) The string "DSK1." followed by the program name, followed with zero.

The code was kept to a minimum, in hopes of clarifing the essentials. Line 10 clears the screen, sets up B$ to hold the names of the files to be loaded from the disk, and opens a file named "DSK1." recognizes it as being a disk directory. Then it inputs a string and three values to move the record pointer up to read the first file on the disk. Finally we set R=1, and start a FOR-NEXT loop of Y to load B$ with the program names to RUN

Lines 20 to 30 completes the loop of Y. The only values needed by the routine , are the program name, and the filetype and the filetype is put into A. We then read two more values into Z to move the pointer to the start of next record. If the length of B$(R) is equal to zero, we have run out of records, and GOTO line 40 because our list is complete. If we we still have records to read, we check A for a filetype of 5. If A= 5, we have the name of a BASIC type program file in B$(R), so it is displayed on the screen, along with the present value of R to be selected by it's number). Since we want to keep this name, we add 1 to R. This will count the names in B$, completely seperate of the value of Y. If A is not equal to 5, we discard that record by dropping down to line 30 and repeat Y's FOR-NEXT loop.

Line 40 is the exit routine for our Y loop. It closes the directory file, proats the selection of Y, which is the number of the program to run from the list. Next we peek into -31950 to see where the last line in our program is at in memory. Since we have two bytes in A and B, we convert them into one value with A=A 6 +B. This number has to be inverted to adjust for two's complement arithmetic, so we subtract 65535. (it should be 65536 but i added a -1 to skip over the code length byte.

To avoid repeated calculation of the length of B$, line 50 sets LN equal to the length just once. Then we can start building the line of code starting at the location we have in A. The breakdown of the bytes being loaded are:

1.3+LN-total length of the command line
2.169 -code token for RUN
3.199 -code token for " (open quote)
4.LN+5-length of string (5="DSK1.")
5.68 -character for letter "D"
6.83 -character for letter "S"

5

7.75 -character for letter "K"
8.49 -character for letter "1"
9.46 -character for letter "."

The characters of the program name are loaded by line 60 by transferring them directly from the screen with GCHAR to our new line of code with CALL LOAD within a FOR-NEXT loop. After the loop ends, a zero is added to the end.

At this time in the running of the program, line 70 no longer looks like it does now. If we were to select "LOAD" from the list of programs, line 70 would contain the following decimal bytes: 13,169,199,9,68,83,75,49,46,76,79,65,68 Which translates to: RUN"DSK1.LOAD". The REM statement at line 70 is only needed to reserve space for the code that is going to be placed there. This program can be modified and still work only if line 70 is re-entered last. A quick way to do this is type 70, then fctn X, then ENTER, then REDO. Press ENTER once more and the line has been re-entered. It is now the last line of code in the Program Space, and pointed to by -31950. If you do not re-enter the line, the new RUN statement will be written into the last line entered. (crash!)

I hope you have enjoyed this little side trip, and found it useful, not only in loading unknown programs, but also in a stepping stone into some of the hidden mysteries buried under the cover of one of the worlds best home computers. Any questions or comments about this series or suggestions for future articles are welcome and could be addressed to either

St. Louis 99ers	CAVE MAN
P.O. BOX 260326	2848 Ohio
CRESTWOOD, MO.	St. Louis, Mo.

And dont forget to share and enjoy.

```
**********************
*                    *
*   NOW YOU CAN      *
*                    *
* MAKE YOUR OWN      *
*                    *
* CUSTOM LABELS      *
*                \   *
* ---------------->  *
*                /   *
**********************
```

```
%%%%%%%%%%%%%%%%
% EASY LOADER %
%   PROGRAM    %
%%%%%%%%%%%%%%%%
10 CALL CLEAR :: CALL INIT :
: DIM B$(127):: OPEN #1:"DSK
1.",INPUT ,INTERNAL,RELATIVE
 :: INPUT #1:A$,X,X,X :: R=1
 :: FOR Y=1 TO 127
20 INPUT #1:B$(R),A,Z,Z :: I
F LEN(B$(R))=0 THEN 40 ELSE
IF ABS(A)=5 THEN DISPLAY AT(
R,1):R;TAB(5);B$(R):: R=R+1
30 NEXT Y
40 CLOSE #1 :: DISPLAY AT(24
,1):"SELECT ONE " :: ACCEPT
AT(24,12):Y :: CALL PEEK(-31
950,A,B):: A=A*256+B-65535
50 LN=LEN(B$(Y)):: CALL LOAD
(A,9+LN,169,199,LN+5,68,83,7
5,49,46)
60 FOR X=1 TO LN :: CALL GCH
AR(Y,X+6,Z):: CALL LOAD(A+X+
8,Z):: NEXT X :: CALL LOAD(A
+X+8,0)
70 ! RE-ENTER THIS LINE LAST

***************
* LABLE MAKER *
***************
   THIS LITTLE PROGRAM FIRST
APPEARED IN FAMILY COMPUTING
MAGAZINE. PROGRAM REVISIONS
HAVE BEEN MADE BY GIL PAULS.
10 DIM L$(5)
20 SP$=CHR$(32)
30 OS$=CHR$(45)
40 FOR I=2 TO 30
50 SP$=SP$&CHR$(32)
60 OS$=OS$&CHR$(45)
70 NEXT I
80 CALL CLEAR
90 PRINT TAB(5);"ALL-PURPOSE
 LABELMAKER"
100 PRINT
110 PRINT "YOU CAN PRINT UP
TO FIVE"
120 PRINT "LINES ON EACH LAB
EL."
130 PRINT
140 PRINT "FOR EACH LINE, TY
PE UP TO"
150 PRINT "30 CHARACTERS AND
 PRESS"
160 PRINT "(ENTER); OR JUST
PRESS"
170 PRINT "(ENTER) WHEN DONE
."
180 PRINT
190 LN=1
200 PRINT "LINE #";CHR$(LN+4
8);
210 INPUT L$(LN)
220 IF L$(LN)="" THEN 310
230 IF LEN(L$(LN))(=30 THEN
290
240 PRINT
250 PRINT "THAT LINE WAS TOO
 LONG."
260 PRINT "PLEASE TRY AGAIN.
"
270 PRINT
280 GOTO 200
290 LN=LN+1
300 IF LN(6 THEN 200
310 CALL CLEAR
320 PRINT "DO YOU WANT THE L
ABEL(S)"
330 PRINT
340 PRINT "(C)ENTERED, OR"
350 PRINT "(L)EFT-JUSTIFIED"
360 CALL KEY(3,CN,S)
370 IF (CN()67)+(CN()99)+(CN
()76)+(CN()108)=-4 THEN 360
380 PRINT
390 INPUT "HOW MANY LABELS?
":NL
400 IF NL(1 THEN 390
410 PRINT "WHEN YOUR PRINTER
 IS READY,"
420 PRINT "PLEASE PRESS ANY
KEY."
430 CALL KEY(3,K,S)
440 IF S(1 THEN 430
450 OPEN #1:"RS232/2.BA=2400
"
460 FOR J=1 TO NL
470 FOR I=1 TO 6
480 IF I(LN THEN 510
490 PRINT #1
500 GOTO 540
510 IF (CN=76)+(CN=108)THEN
530
520 PRINT #1:SEG$(SP$,1,(15-
LEN(L$(I))/2));
530 PRINT #1:L$(I)
540 NEXT I
550 NEXT J
560 CLOSE #1
570 CALL CLEAR
580 PRINT "WOULD YOU LIKE TO
"
590 PRINT
600 PRINT "(P)RINT THIS LABE
L AGAIN,"
610 PRINT "(C)OMPOSE ANOTHER
 LABEL, OR"
620 PRINT "(Q)UIT?"
630 CALL KEY(3,K,S)
640 IF (K=80)+(K=112)THEN 31
0
650 IF (K=67)+(K=99)THEN 80
660 IF (K=81)+(K=112)=0 THEN
 630
670 END
```

****
* The Game of Black Box *
* Steve Karasek *
****

Computers are often called "black boxes", because people use them, putting data in and getting information out, without really knowing what's inside. In this game, you are presented with a black box, and the object is to determine what's inside, or more appropriately, where.

The box is divided into an 8 by 8 grid, labelled on all four sides with 32 letters and numbers. By pressing one of the corresponding keys, a probe is sent into the box at that point. Inside the box are four obstacles, each occupying one grid space. If the probe hits one of these obstacles head-on, it will bounce back (a 180-degree turn). If it encounters a probe in a grid space diagonal to it, it will deflect away at a 90-degree turn. The object of the game is to determine where the four obstacles are by observing where each probe exits the box.

For example, the left side of the box is labelled with A through H, and the top is labelled with 1 through 8. If you press A, another A will appear next to the box at the top of the left hand side. This marks the entry point of the probe. If there are no obstacles in the first two rows of the box, the probe will pass straight through the top row, emerging at the top of the right hand side. Another A will appear there to mark the exit point. If there is an obstacle in the second row, say point B5 for example, the obstacle will travel along the top until it gets to point A4. There will then be an obstacle next to the probe diagonally, so it will be deflected and shoot straight up, emerging next to the label 4 at the top.

If there is an obstacle in the first row, the probe will turn around and exit at the same point that it entered. When this happens, the number or letter at the entry point will be replaced by an asterisk (*), which signifies that the entry point and the exit point are the same. An obstacle at point B1 will also cause an asterisk, since the probe will try to deflect upward before it even gets into the box. This obstacle will cause an asterisk at entry points A, B, and C.

Of course, a probe may come in contact with more than one obstacle before it exits. It often will deflect off one obstacle, hit another one head-on, come back and deflect off the first one again, and exit at the entry point. With the right arrangement of obstacles, it is possible that a probe will appear to pass straight through without hitting any obstacles, when in fact it has deflected off of all four!

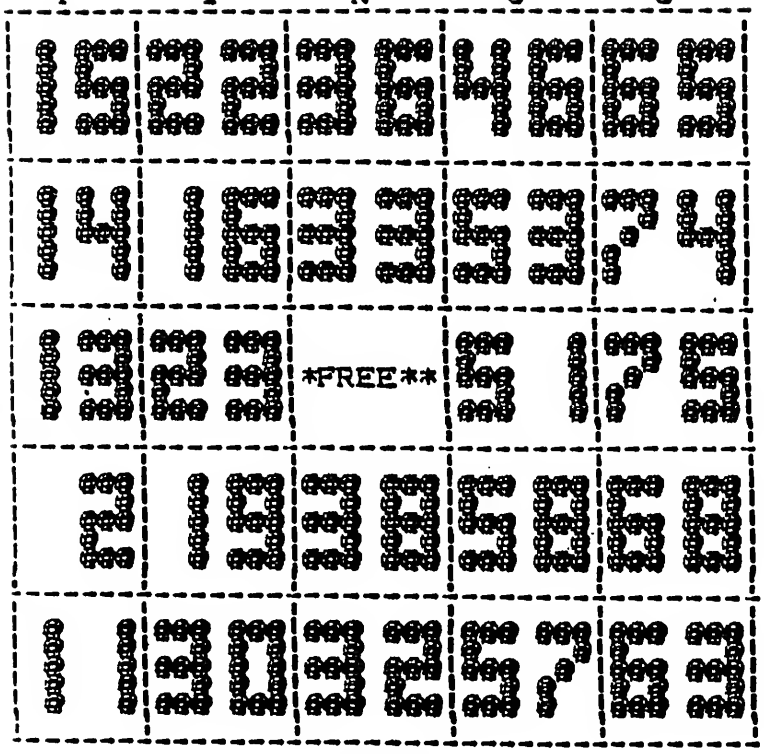
When you think you know where an obstacle is, press the space bar. You will be prompted "GUESS?". Enter a letter A through H and a number 1 through 8 specifying the grid space (for example, type B4 followed by the "enter" key). If you guessed correctly, the obstacle will appear on the box. The object is to find all four obstacles in as few moves as possible. If you want to give up, just type a question mark in response to the "GUESS?" prompt, and all of the obstacles will be displayed.

Your score is displayed in the upper right hand corner of the screen. Each time you fire a probe or make an INCORRECT guess, 1 is added to your score. A score of around 10 is pretty good, 5 is excellent. Some of the more tricky layouts may raise your score to around 20.

To get an idea of how the probes react in different situations, I suggest that you start out by taking 5 to 10 pot shots at the box, then typing a question mark in response to the "GUESS?" prompt to see where the obstacles are, and following the path of the probes that you sent in. If the probes don't seem to be reacting the way they should, list the program and check for typing errors.

**** BLACK BOX PROGRAM ****

```
100 DISPLAY AT(7,10)ERASE AL
L:"BLACK BOX" :: RANDOMIZE :
: CALL MAGNIFY(2)
110 X0=7 :: Y0=3 :: NBLOCK=4
 :: DIM S$(4),B(9,9),USED(8,
8),BLOCK(4,1):: DEF R=INT(RN
D*8)+1
120 FOR I=0 TO 4 :: READ S$(
I):: NEXT I
130 DATA 00000000,08080808,0
F,F8,FF
140 C=91 :: FOR I=0 TO 1 ::
FOR K=0 TO 1 :: FOR J=2 TO 4
 :: CALL CHAR(C,S$(I)&S$(J)&
S$(K)):: C=C+1 :: NEXT J ::
NEXT K :: NEXT I
150 CALL CHAR(C,S$(1)&S$(1))
:: CALL CHAR(104,"FCFCFCFCFC
FC")
160 FOR I=1 TO 8 :: CALL HCH
AR(Y0-2,X0+I+1,48+I):: CALL
HCHAR(Y0+I+1,X0-2,64+I)
170 CALL HCHAR(Y0+20,X0+I+1,
72+I):: CALL HCHAR(Y0+I+1,X0
+20,80+I):: NEXT I
180 CALL OUT(1,94,93,96,95):
: FOR I=2 TO 16 STEP 2 :: CA
LL OUT(I,103,32,103,103):: I
F I<16 THEN CALL OUT(I+1,100
,93,102,101)
190 NEXT I :: CALL OUT(17,97
,93,99,98)
200 FOR I=1 TO 8 :: FOR J=1
TO 8 :: USED(I,J)=0 :: NEXT
J :: NEXT I :: NTRY=-1 :: GO
SUB 550 :: NC=0
210 FOR I=1 TO NBLOCK :: R1=
R :: R2=R :: IF B(R1,R2)THEN
 210
220 BLOCK(I,0)=R1 :: BLOCK(I
,1)=R2 :: B(R1,R2)=1 :: NEXT
 I
230 CALL KEY(0,T,S):: IF S<>
1 THEN 230 ELSE IF T=32 THEN
 390
240 X=T-48 :: IF X>0 AND X<9
 THEN Y,DX=0 :: DY=1 :: GOTO
 280
250 Y=T-64 :: IF Y>0 AND Y<9
 THEN X,DY=0 :: DX=1 :: GOTO
 280
260 X=T-72 :: IF X>0 AND X<9
 THEN Y=9 :: DX=0 :: DY=-1 :
: GOTO 280
270 Y=T-80 :: IF Y>0 AND Y<9
 THEN X=9 :: DY=0 :: DX=-1 E
LSE 230
280 CALL GCHAR(Y0+Y+Y,X0+X+X
,S):: IF S<>32 THEN 230 ELSE
 GOSUB 550
290 CALL HCHAR(Y0+Y+Y,X0+X+X
,T):: SX=X :: SY=Y
300 IF B(X+DX,Y+DY)OR B(X+DX
+DY,Y+DX+DY)OR B(X+DX-DY,Y+D
Y-DX)THEN GOSUB 540 :: CALL
HCHAR(Y0+Y+Y,X0+X+X,ASC("*")
):: GOTO 230
310 IF B(X+DX,Y+DY)THEN DX=-
DX :: DY=-DY :: GOTO 310
320 IF DY=-1 OR DX=-1 THEN I
F B(X-1,Y-1)THEN DY=DY+1 ::
DX=DX+1
330 IF DY=-1 OR DX=1 THEN IF
 B(X+1,Y-1)THEN DY=DY+1 :: D
X=DX-1 :: GOTO 310
340 IF DY=1 OR DX=1 THEN IF
B(X+1,Y+1)THEN DY=DY-1 :: DX
=DX-1 :: GOTO 310
350 IF DY=1 OR DX=-1 THEN IF
 B(X-1,Y+1)THEN DY=DY-1 :: D
X=DX+1 :: GOTO 310
360 X=X+DX :: Y=Y+DY :: IF X
>0 AND X<9 AND Y>0 AND Y<9 T
HEN 310
370 IF X=SX AND Y=SY THEN T=
ASC("*")
380 CALL HCHAR(Y0+Y+Y,X0+X+X
,T):: GOTO 230
390 DISPLAY AT(24,1):"GUESS?
" :: ACCEPT AT(24,8):X$ :: I
F X$="?" THEN 490 ELSE IF LE
N(X$)=2 THEN 410
400 CALL SOUND(120,220,0)::
GOTO 450
410 S=ASC(X$)-64 :: IF S<1 O
R S>8 THEN 400 ELSE T=ASC(SE
G$(X$,2,1))-48 :: IF T<1 OR
T>8 THEN 400
420 IF USED(T,S)THEN DISPLAY
 AT(24,12):"ALREADY GUESSED"
 :: GOTO 400
430 USED(T,S)=1 :: IF B(T,S)
THEN 460
440 GOSUB 550 :: DISPLAY AT(
24,12):"INCORRECT" :: CALL S
OUND(120,110,0)
450 GOSUB 530 :: DISPLAY AT(
24,1):: GOTO 230
460 NC=NC+1 :: CALL SPRITE(#
NC,104,7,(Y0+S+S)*8-9,(X0+T+
470 IF NC<NBLOCK THEN DISPLA
Y AT(24,12):"CORRECT" :: CAL
L SOUND(120,1400,0):: GOTO 4
50
480 DISPLAY AT(24,1):"     C
 O R R E C T !" :: CALL SOUN
D(1000,262,0,330,0,392,0)::
GOSUB 530 :: GOTO 500
490 FOR I=1 TO 4 :: CALL SPR
ITE(#I,104,7,(Y0+2*BLOCK(I,1
))*8-9,(X0+2*BLOCK(I,0))*8-9
):: NEXT I :: NTRY=98 :: GOS
UB 550
500 DISPLAY AT(24,1):"PRESS
<ENTER> FOR NEW GAME" :: ACC
EPT AT(24,28):X$ :: DISPLAY
AT(24,1)
510 CALL HCHAR(Y0,X0+2,32,15
):: CALL VCHAR(Y0+2,X0,32,15
):: CALL HCHAR(Y0+18,X0+2,32
,15):: CALL VCHAR(Y0+2,X0+18
,32,15)
520 CALL DELSPRITE(ALL):: FO
R I=1 TO 4 :: B(BLOCK(I,0),B
LOCK(I,1))=0 :: NEXT I :: GO
TO 200
530 FOR I=1 TO 500 :: NEXT I
 :: RETURN
540 FOR I=1 TO 100 :: NEXT I
 :: RETURN
550 NTRY=NTRY+1 :: DISPLAY A
T(1,26):NTRY :: RETURN
560 SUB OUT(I,A,B,C,D):: DIS
PLAY AT(3+I,6)SIZE(17):CHR$(
A)&RPT$(CHR$(B)&CHR$(C),7)&C
HR$(B)&CHR$(D):: SUBEND
```



TIngo
by Steve Karasek

TIngo (or TI-Bingo) will print Bingo cards and call the game for you. It starts out by asking how many cards it should print (you can re-use the same cards when you play later, or you can use your own cards if you have them). It will print them 4 to a page. These should be cut and glued or taped to cardboard to make them sturdier. It takes a while to print the cards, so get it started early, before you're actually ready to play. Pennies or buttons can be used as markers.

When everyone has their cards and markers, press the ENTER key as instructed to start the game. The computer will display the letters and numbers on the screen in large letters and also say them if you have a speech synthesizer attached. It will also display on the screen all of the numbers called so far. When someone has a TINGO or if you want to pause it to check the numbers, press any key. You will be instructed to press C to continue or N for a new game (make sure the ALPHA LOCK key is depressed). If there was a TINGO and you're ready for a new game, press N, otherwise press C to continue the current game.

If you want to vary the speed at which the numbers are called, change the 150 in line 410 to a larger number for slower speed or a smaller number for faster speed.

If your printer is not named "PIO", change the name in line 140. The last part of this line sets the printer line spacing to 7/72 inch. If you do not have an EPSON-compatible printer, you will have to change this to the codes needed by your printer to set the line spacing. If you can't set it to 7/72 inch, set it to 8 or 10 lines per inch.

The digits following the exclamation points at the end of each line are checksums for the CHECKSUM program, which appeared in an earlier newsletter. Do not type these characters. Use the CHECKSUM program to guard against typos.

```
**********************
* TIngo - the program *
**********************

100 L$=RPT$("-",80):: M$="!
    "&RPT$("!       ",4)&"!
" :: M$=M$&M$ !057
110 CALL MAGNIFY(2):: RANDOM
IZE :: DIM U(75,1),T$(7),U$(
15):: FOR I=0 TO 9 :: READ P
(I):: NEXT I !073
120 DATA 31599,18724,29671,3
1207,18925,31183,31695,4775,
31727,31215 !222
130 DISPLAY ERASE ALL AT(8,1
2):"TIngo" :: DISPLAY AT(10,
7):"By Steve Karasek" !204
140 INPUT "HOW MANY CARDS TO
 PRINT? ":N :: IF N THEN OPE
N #1:"PIO",OUTPUT,VARIABLE 2
55 :: PRINT #1:CHR$(27);"A";
CHR$(7);!116
150 FOR I=1 TO (N+1)/2 :: PR
INT #1 :: FOR J=1 TO 10 :: P
RINT #1:TAB(J*8-4);SEG$("TIN
GOTINGO",J,1);:: NEXT J :: P
RINT #1 !021
160 FOR M=1 TO 5 :: PRINT #1
:L$:M$ :: FOR M=0 TO 1 :: FO
R J=0 TO 4 !021
170 K=INT(RND*15)+1+J*15 ::
IF U(K,M)THEN 170 !254
180 C(J,M)=K :: U(K,M)=1 ::
NEXT J :: NEXT M :: FOR K=0
TO 4 :: FOR M=0 TO 1 :: FOR
J=0 TO 4 :: N$=N$&"!" :: IF
M()3 OR J()2 THEN 210 !251
190 IF K=2 THEN N$=N$&"*FREE
**" ELSE N$=N$&"      " !13
7
200 GOTO 250 !073
210 X=0 :: FOR W=1 TO 0 STEP
 -1 :: X=INT(C(J,M)/10^W)-X*
10 !037
220 FOR L=0-(J=0 AND W=1)TO
2 :: IF (P(X)AND 2^(L+K*3)))
0 AND(W=0 OR X)0)THEN N$=N$&
"#" ELSE N$=N$&" " !046
230 NEXT L :: IF W THEN N$=N
$&" " !049
240 NEXT W !237
250 NEXT J :: N$=N$&"!" :: N
EXT M :: FOR M=1 TO LEN(N$):
: IF SEG$(N$,M,1)="#" THEN P
RINT #1:"@";ELSE PRINT #1:"
";!009
260 NEXT M :: PRINT #1:CHR$(
13);N$ :: N$="" :: NEXT K ::
 PRINT #1:M$ :: NEXT M :: PR
INT #1:L$ !245
270 IF INT(I/2)*2=I THEN PRI
NT #1:CHR$(12);!169
280 GOSUB 470 :: NEXT I :: I
F N THEN CLOSE #1 !035
290 FOR I=2 TO 7 :: READ T$(
I):: NEXT I :: FOR I=1 TO 15
 :: READ U$(1):: NEXT I !128
300 DATA TWENTY,THIRTY,FORTY
,FIFTY,SIXTY,SEVENTY !152
310 DATA ONE,TWO,THREE,FOUR,
FIVE,SIX,SEVEN,EIGHT,NINE,TE
N,ELEVEN,TWELVE,THIRTEEN,FOU
RTEEN,FIFTEEN !114
320 CALL CLEAR :: FOR I=9 TO
 14 :: READ J :: CALL COLOR(
I,J,J):: NEXT I !167
330 DATA 6,7,13,5,14,3 !104
340 Z=0 :: CALL DELSPRITE(AL
L):: INPUT "PRESS (ENTER) WH
EN READY ":X$ :: DISPLAY ERA
SE ALL AT(1,5):"T   I   N
   G   O" !240
350 J=4 :: FOR I=96 TO 136 S
TEP 8 :: CALL VCHAR(1,J,I,17
):: J=J+5 :: NEXT I !181
360 IF Z=75 THEN 460 !024
370 J=INT(RND*75):: IF U(J,0
)THEN 370 !190
380 Z=Z+1 :: U(J,0)=1 :: I=I
NT(J/15):: J=J+1 :: DISPLAY
AT(J-I*15+2,4+I*5)SIZE(2):US
ING ("##"):J :: X$=SEG$("TIN
GO",I+1,1)!211
390 CALL SPRITE(#4,ASC(X$),2
,144,104):: Y$=STR$(J)&" " :
: FOR I=1 TO LEN(Y$):: CALL
SPRITE(#I,ASC(SEG$(Y$,I,1)),
2,144,114+I*14):: NEXT I !12
3
400 CALL SAY(X$):: IF J)15 A
ND J(20 THEN CALL SAY(U$(J-1
0),,"TEEN")ELSE X=INT(J/10)*
-(J)19):: CALL SAY(T$(X),,U$
(J-X*10))!181
410 FOR I=1 TO 150 :: CALL K
EY(0,X,S):: IF S THEN 430 !1
06
420 NEXT I !223
430 IF S=0 THEN 360 ELSE DIS
PLAY AT(22,1):"PRESS C TO CO
NTINUE OR N FORA NEW GAME" !
096
440 CALL KEY(0,X,S):: IF X=-
1 THEN 440 !210
450 X$=CHR$(X):: IF X$="C" T
HEN CALL HCHAR(22,1,32,64)::
 GOTO 360 ELSE IF X$()"N" TH
EN 440 !051
460 GOSUB 470 :: GOTO 340 !0
77
470 FOR J=0 TO 75 :: U(J,0),
U(J,1)=0 :: NEXT J :: RETURN
 !061
```

LOGIX

By Rich Renth

This program was written in response to a request for a "Mastermind" type program. The object is to put the correct colors in the proper order. Instructions are in the program. This is a challenging game for all ages. Enjoy!! (Editors Note: This program is available in the club library on disk and cassette. Thanks Rich!)

```
110 CALL CLEAR
120 CALL SCREEN(8)
130 PRINT "(C)olor or (B)lac
k & white"
140 INPUT "ENTER YOUR LETTER
 CHOICE )":AN$
150 IF AN$="C" THEN 170
160 CALL SCREEN(16)
170 CALL CLEAR
180 PRINT TAB(10);"L O G I X
": :"the object of the game
is toguess the proper or
der and color of the four p
egs that"
190 PRINT "the computer will
 hide underthe question mark
s at the top. the fou
r pegs are all a different c
olor, picked"
200 PRINT "from the six colo
rs. the computer will hel
p you each time you ente
r your four color guesses
, by telling"
210 PRINT "you just how many
 colors areright and how man
y of them are in the ri
ght row. you can have up t
o ten attempts"
220 PRINT "to guess the prop
er order and color of the
hidden pegs": : :" PRESS
 ANY KEY TO START GAME"
230 CALL KEY(0,K,S)
240 IF S<1 THEN 230
250 DATA 00000000FF,00000000
FF10101,00000000F010101,0000
00001F10101,10101010F,10
1010101F
260 DATA 10101010FF10101,101
010101010101,10101010FF,1010
10101F10101,10101010F010
101,FF81BDA5A5BD81FF
270 DATA 9,11,3,14,16,6
280 DATA 0078444478504844,00
44442810101010,003C40405C444
438,0044444428281010,004
4444454545428,00782424382424
78
290 FOR X=35 TO 46
300 READ A$
310 CALL CHAR(X,A$)
320 NEXT X
330 IF AN$="B" THEN 370
340 FOR X=96 TO 136 STEP 8
350 CALL CHAR(X,"FFFFFFFFFFFF
FFFF")
360 NEXT X
370 FOR X=9 TO 14
380 READ Y
390 CALL COLOR(X,Y,1)
400 NEXT X
410 IF AN$="C" THEN 470
420 FOR X=96 TO 136 STEP 8
430 READ A$
440 CALL COLOR(X/8-3,2,1)
450 CALL CHAR(X,A$)
460 NEXT X
470 RANDOMIZE
480 FOR X=1 TO 4
490 A(X)=INT(RND*6+1)
500 FOR Y=1 TO X-1
510 IF A(X)=A(Y)THEN 490
520 NEXT Y
530 NEXT X
540 CALL CLEAR
550 PRINT TAB(11);"&&$$$$$$%
  RIGHT"
560 PRINT CHR$(96);"R ED";TA
B(11);"$ $ $ $ $"
570 PRINT CHR$(136);"B LUE";
TAB(11);"(8+8+8+8" COL  ROW"
580 PRINT CHR$(128);"W HITE"
;TAB(11);"&&$$$$$%"
590 PRINT CHR$(112);"G REEN"
;TAB(11);"$ $ $ $ $"
600 PRINT CHR$(120);"V IOLET
";TAB(11);",$)$)$)$-"
610 PRINT CHR$(104);"Y ELLOW
";TAB(11);"$ $ $ $ $"
620 PRINT TAB(11);",$)$)$)$-
"
630 PRINT TAB(11);"$ $ $ $ $
"
640 PRINT TAB(11);",$)$)$)$-
"
650 PRINT TAB(11);"$ $ $ $ $
"
660 PRINT TAB(11);",$)$)$)$-
"
670 PRINT TAB(11);"$ $ $ $ $
"
680 PRINT TAB(11);",$)$)$)$-
"
690 PRINT TAB(11);"$ $ $ $ $
"
700 PRINT TAB(11);",$)$)$)$-
"
710 PRINT TAB(11);"$ $ $ $ $
"
720 PRINT TAB(11);",$)$)$)$-
"
730 PRINT TAB(11);"$ $ $ $ $
"
740 PRINT TAB(11);",$)$)$)$-
"
750 PRINT TAB(11);"$ $ $ $ $
"
760 PRINT TAB(11);",$)$)$)$-
"
770 PRINT " COLOR?";TAB(11);
"$ $ $ $ $"
780 PRINT TAB(11);"(8+8+8+8"
";
790 IF AN$="C" THEN 810
800 CALL VCHAR(2,3,32,6)
810 FOR C=14 TO 20 STEP 2
820 FOR R=5 TO 23 STEP 2
830 CALL HCHAR(R,C,46)
840 NEXT R
850 NEXT C
860 FOR C=14 TO 20 STEP 2
870 CALL HCHAR(2,C,63)
880 NEXT C
890 R=23
900 W=0
910 B=0
920 FOR C=14 TO 20 STEP 2
930 GOSUB 1080
940 CALL HCHAR(R,C,K*8+88)
950 IF A(C/2-6)<>K THEN 970
960 B=B+1
970 FOR X=1 TO 4
980 IF A(X)<>K THEN 1000
990 W=W+1
1000 NEXT X
1010 NEXT C
1020 CALL HCHAR(R,24,W+48)
1030 CALL HCHAR(R,29,B+48)
1040 R=R-2
1050 IF B=4 THEN 1340
1060 IF R<5 THEN 1340
1070 GOTO 900
1080 CALL HCHAR(R,C,88)
1090 CALL HCHAR(23,10,95)
1100 CALL KEY(0,K,S)
1110 CALL HCHAR(R,C,32)
1120 CALL HCHAR(23,10,32)
1130 IF S<1 THEN 1080
1140 CALL HCHAR(23,10,K)
1150 IF (K=82)+(K=89)+(K=71)
+(K=86)+(K=87)+(K=66)THEN 11
90
1160 CALL SOUND(-50,220,0)
1170 CALL SOUND(250,110,0)
1180 GOTO 1080
1190 CALL SOUND(-50,880,0)
1200 CALL SOUND(-50,988,4)
1210 IF K<>82 THEN 1230
1220 K=1
1230 IF K<>89 THEN 1250
1240 K=2
1250 IF K<>71 THEN 1270
1260 K=3
1270 IF K<>86 THEN 1290
1280 K=4
1290 IF K<>87 THEN 1310
1300 K=5
1310 IF K<>66 THEN 1330
1320 K=6
1330 RETURN
1340 FOR X=1 TO 4
1350 CALL HCHAR(2,X*2+12,A(X
)*8+88)
1360 NEXT X
1370 L=11
1380 M$=" WELL YOU "
1390 GOSUB 1700
1400 IF B<4 THEN 1480
1410 M$=" MADE IT"
1420 GOSUB 1700
1430 M$=" IN ONLY"
1440 GOSUB 1700
1450 M$=" "&STR$(ABS((R+1)/2
-12))&" TRIES"
1460 GOSUB 1700
1470 GOTO 1540
1480 M$="MIGHT MAKE"
1490 GOSUB 1700
1500 M$=" IT NEXT"
1510 GOSUB 1700
1520 M$=" TIME"
1530 GOSUB 1700
1540 L=L+2
1550 M$=" PLAY"
1560 GOSUB 1700
1570 M$=" AGAIN"
1580 GOSUB 1700
1590 M$=" Y/N?"
1600 L=L+1
1610 CALL KEY(0,K,S)
1620 CALL HCHAR(20,5,32)
1630 CALL HCHAR(20,7,32)
1640 GOSUB 1700
1650 L=20
1660 IF S<1 THEN 1610
1670 IF K=89 THEN 470
1680 IF K<>78 THEN 1610
1690 END
1700 FOR X=1 TO LEN(M$)
1710 C=ASC(SEG$(M$,X,1))
1720 CALL HCHAR(L,X+2,C)
1730 NEXT X
1740 L=L+1
1750 RETURN
```

Maze Maker
by Steve Karasek

This program will print mazes for you to solve. It asks for the number of mazes to print, then for the level of difficulty, from 0 to 9. Level 0 is a VERY trivial maze (a child's first maze, perhaps), while level 9 is fairly challenging. The level number is printed at the top of the maze.
No matter what level you select, the maze will be printed to fill as much of the page as possible, so the lower-level mazes will have wider pathways which are easier for young children. There will always be exactly one path from Start to Finish.
The higher-level mazes take a while to compute. In particular, level 9 mazes take over 20 minutes each. You can always start up the program and come back a few hours later. The program keeps track of how far it has gone in computing each maze by displaying a line of the form M / N on the screen, where N is the number of squares in the maze and M is the number of squares the program has computed a path to. When M equals N, the maze is done and is sent to the printer.
If your printer is not named "PIO", change the name in line 110. The last part of this line sets the printer line spacing to 7/72 inch. If you do not have an EPSON-compatible printer, you will have to change this to the codes needed by your printer to set the line spacing. If you can't set it to 7/72 inch, set it to 8 or (preferably) 10 lines per inch.
The !'s and numbers at the end of each line are the checksums for Tom Freeman's CHECKSUM program, and are not needed by the maze program.

```
**********************
* MAZE - THE PROGRAM *
**********************

100 RANDOMIZE :: OPTION BASE
 1 :: DIM M(39,39):: INPUT "
HOW MANY MAZES? ":Z :: PRINT
 !223
110 INPUT "LEVEL OF DIFFICUL
TY(0-9)? ":L :: IF L<0 OR L>
9 THEN 110 ELSE OPEN #1:"PIO
",OUTPUT :: PRINT #1:CHR$(27
);"A";CHR$(7);!131
120 N=INT(L+1)*4+(L=4 OR L=9
):: X=80/N :: S=INT(X):: S=S
+(X=S)!138
130 PRINT #1:"Start";TAB(30)
;"Level";L :: FOR X=1 TO N ::
 FOR Y=1 TO N :: M(X,Y)=0 ::
 NEXT Y :: NEXT X :: IF N=39
 THEN 150 !174
140 FOR X=1 TO N :: M(N+1,X)
,M(X,N+1)=16 :: NEXT X !203
150 C,X,Y=1 :: DISPLAY ERASE
 ALL AT(12,12):"1 /";N*N ::
 ON ERROR 290 !059
160 W=INT(RND*4):: DX=X+(W=0
)-(W=1):: DY=Y+(W=2)-(W=3)::
 K=M(DX,DY):: IF K THEN
 160 !229
170 M(X,Y)=M(X,Y)+2^W :: IF
 INT(W/2)*2=W THEN W=W+1 ELSE
 W=W-1 !125
180 X=DX :: Y=DY :: M(X,Y)=M
(X,Y)+2^W :: C=C+1 :: DISPLA
Y AT(12,9)SIZE(4):USING "###
#":C :: IF C=N*N THEN 240 !0
53
190 IF X<N THEN IF M(X+1,Y)=
0 THEN 160 !198
200 IF Y<N THEN IF M(X,Y+1)=
0 THEN 160 !199
210 IF Y>1 THEN IF M(X,Y-1)=
0 THEN 160 !117
220 IF X>1 THEN IF M(X-1,Y)=
0 THEN 160 !116
230 X=INT(RND*N)+1 :: Y=INT(
RND*N)+1 :: IF M(X,Y)THEN 19
0 ELSE 230 !248
240 ON ERROR STOP :: PRINT #
1 :: PRINT #1:"#";TAB(S+1);R
PT$("#",S*(N-1)+1):: S=S
-1 :: S$=RPT$(" ",S):: X$=RP
T$("#",S)!069
250 M(N,N)=M(N,N)+8 :: FOR Y
=1 TO N :: FOR W=1 TO S :: P
RINT #1:"#";:: FOR X=1 TO N
 :: PRINT #1:S$;!076
260 IF M(X,Y)AND 2 THEN PRIN
T #1:" ";ELSE PRINT #1:"#";!
084
270 NEXT X :: PRINT #1 :: NE
XT W :: PRINT #1:"#";:: FOR
 X=1 TO N :: IF M(X,Y)AND
 8 THEN PRINT #1:S$;ELSE PRI
NT #1:X$;!244
280 PRINT #1:"#";:: NEXT X :
 : PRINT #1 :: NEXT Y :: S=S+
1 :: PRINT #1: :TAB(S*N-4);"
Finish":CHR$(12);:: Z=Z-1 ::
 IF Z>0 THEN 130 ELSE END !0
20
290 ON ERROR 290 :: RETURN 1
60 !159
```

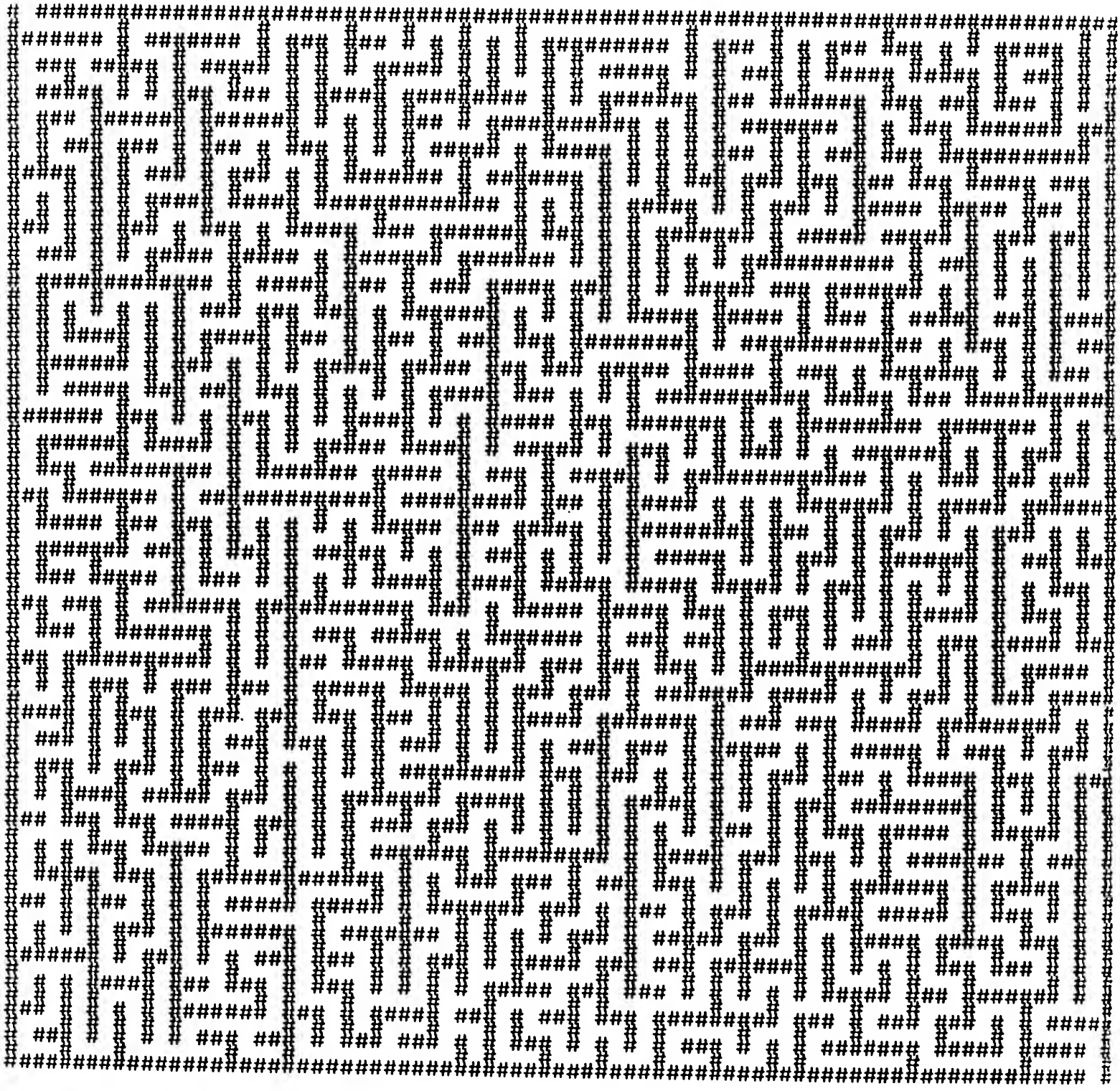
//

EXAMPLE OF A MAZE FROM "MAZE MAKER"

by Steve Karasek

Start

Level 9



Finish

HAPPY HOLIDAYS FROM ST. LOUIS 99'ERS

TI WRITER TABLE SORT
by George Paschetto

12

This program will sort a table written with the TI WRITER. In order to use it, you will need XBASIC.

The program can be very useful for organizing a data base into different indices organized by use, alphabetically, or any other characteristic. Here's an example:

```
1     2                      1     2
  4.87 Photo Album, 3 ring   127.72 Camera, Polaroid
 29.39 Tripod, silk SM300     39.87 Lens, wide angle
 39.87 Lens, wide angle        4.84 Photo Album, 3 ring
127.72 Camera, Polaroid       29.97 Tripod, silk SM300
```

The first example is sorted on field 1, the price. The second is sorted on field 2, the description. I have used this to compare lists of my students- one sorted by age and the other sorted by reading level. I decided to use files created on the TI WRITER because it supports an 80 character line and it allows easy editing of the table. If your table is part of a larger file, use the ability of the TI WRITER to save or load specific lines of a file.

When creating the table for sorting, you must number the fields of your table by placing the number over the first character in that field. The field marking line must be the first in the file, or the program will not be able to find it.

```
CORRECT:                              INCORRECT:
1       2     3                        1       2     3
2801GP    39.87 Lens, wide angle       2801GP    39.87 Lens, wide angle
300PBX    29.97 Tripod, silk SM300     300PBX    29.97 Tripod, silk
610000PL 127.72 Camera, Polaroid       610000PL 127.72 Camera, Polaroid
3050PEB    4.87 Photo Album, 3 ring    3050PEB    4.84 Photo Album, 3 ring
```

The incorrect example has misplaced field markers 1 and 2. The 1 (in 127.72) will not be recognized as part of field number 2, and all the entries in field number 1 will be sorted starting with the second character.

Notice that the 2 marking the second field in the correct example is over the left-most digit of the largest number (the 1 in 127.72). Numbers will not be sorted correctly unless they are right-justified. (You may want to use leading zeroes, 009 010 014 240 etc. for ease, but it is not necessary).

In creating your table, plan ahead so that you know what the largest entry will be. It is perfectly alright to have several blank spaces separating fields.

The file can be as long as 400 lines (not counting the field marker) and each line can have up to 9 fields. The fields must all fit on one line.

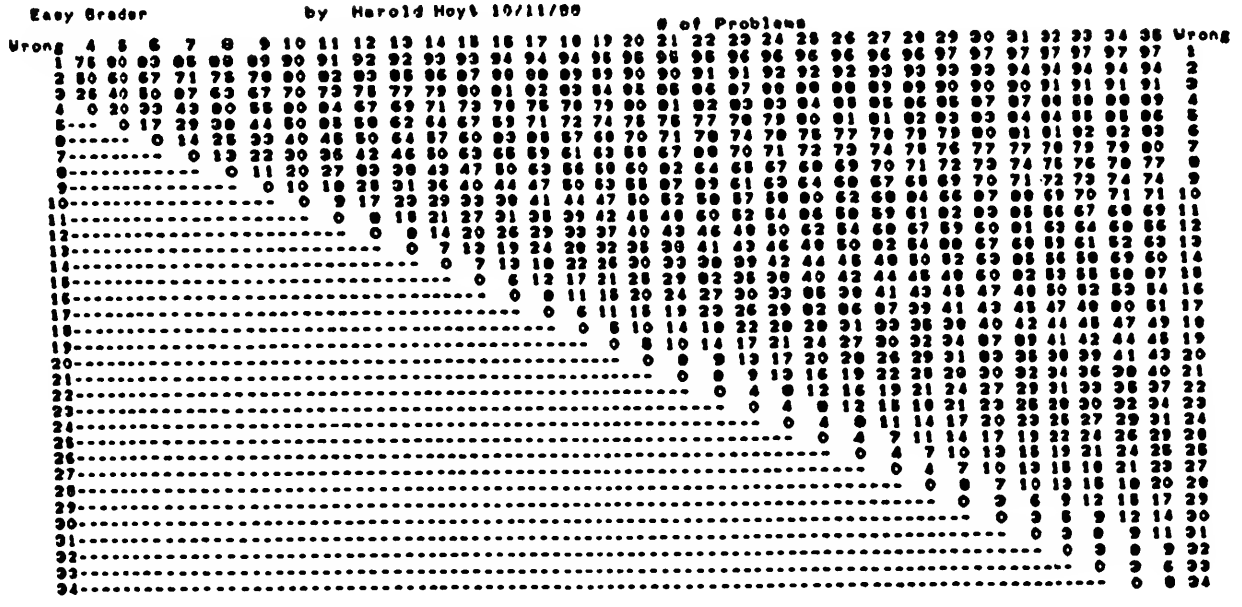
The sort will always be in ascending order (from smallest to largest).

```
100 REM PRESCAN VARIABLES
110 GOTO 120 :: D$,X,Y,K,S,Z
,FN,FS,FL,L,S$,H,J,B$ :: CAL
L KEY :: CALL SOUND :: CALL
HCHAR
120 CALL CLEAR :: DIM A$(400
),F(10)
130 !@P-
140 INPUT "  See instruction
s? Y/N ":D$ :: CALL CLEAR ::
 IF D$="N" THEN 230
150 PRINT " This program was
 written tosort files create
d with the TI WRITER.  The f
ile can be as long as 400 li
nes."
160 PRINT : :" The first lin
e of the file must contain t
he field num- bers, like thi
s:": :
170 PRINT "1   2  3  <-first
 line":"102 NJ PAUL PETERS":
"314 MO CAROL CORRINA":"
622 AL HU NOES": :
180 PRINT " You may have up
to 9 fieldsbut they must all
 fit on oneline.": : :: INPU
T "  (press enter)":D$ :: CA
LL CLEAR
190 PRINT " The lines can be
 up to the full 80 character
s long thatTI WRITER support
s.  This  program will only
 sort one  field at a time."
200 PRINT : :" The sort is a
lways in      ascending orde
r, with the   lowest value f
irst."
210 PRINT : :" The file can
be used by TI WRITER after t
his program  is through wit
h it.": :
220 INPUT "  (press enter)":
D$ :: CALL CLEAR
230 DISPLAY AT(12,1):"Input
device and file name,":"   D
SK"
240 ACCEPT AT(13,7):D$ :: IF
 D$="" THEN 240 ELSE OPEN #1
:"DSK"&D$,DISPLAY ,VARIABLE
80,INPUT
250 CALL CLEAR
260 REM LOOK FOR 1ST LINE
270 LINPUT #1:A$(0):: IF EOF
(1)THEN 600 ELSE IF A$(0)="
" THEN 270
280 FOR X=1 TO 400 :: LINPUT
 #1:A$(X):: IF EOF(1)THEN 31
0
290 NEXT X
300 REM F()=FIELD'S POSITION
S (COUNT DOWN 2 FROM EOF()-L
AST 2 LINES ARE TABS)
310 X=X-2 :: CLOSE #1 :: FOR
 Y=1 TO 9 :: F(Y)=POS(A$(0),
STR$(Y),1):: IF F(Y)=0 THEN
330
320 NEXT Y
330 F(Y)=80 :: Y=Y-1 :: IF Y
 THEN 350
340 PRINT "Can't find field
marker.": :"This was found i
nstead:": :A$(0):: END
350 CALL CLEAR :: DISPLAY AT
(10,5):"Press:":"   <1> Sor
t":"   <2> Save on disk":"
  <3> Quit"
360 CALL KEY(0,K,S):: IF (K<
49)+(K>51)THEN 360 ELSE CALL
 SOUND(-20,880,0):: ON K
-48 GOTO 380,540,580
```

* CONTINUATION OF TI WRITER TABLE SORT *

```
370 REM GET FIELD TO SORT
380 CALL CLEAR :: DISPLAY AT
(1,9):"<choose field)"
390 FOR Z=1 TO Y :: DISPLAY
AT(Z*2,1):"Field#";STR$(Z);"
 ";SEG$(A$(1),F(Z),F(Z+1)-F(
Z)):: NEXT Z
400 DISPLAY AT(20,1):"Sort o
n field number _." :: FN=0
410 CALL KEY(0,K,S):: IF (K=
13)*(FN>0)THEN 440 ELSE IF (
K<49)+(K>48+Y)THEN 410
420 CALL HCHAR(20,24,K):: FN
=K-48 :: GOTO 410
430 REM SORTING (FN=FIELD NI
MBER, FS=FIELD'S START, FL=F
IELD'S LENGTH)
440 CALL CLEAR :: DISPLAY AT
(12,5):"Sorting..." :: FS=F(
FN):: FL=F(FN+1)-FS :: K=X :
: L=INT(X/2)+1
450 IF L<>1 THEN L=L-1 :: S$
=A$(L):: GOTO 470
460 S$=A$(K):: A$(K)=A$(1)::
 K=K-1 :: IF K<1 THEN A$(H)=
S$ :: GOTO 350
470 J=L
480 H=J :: J=J+J :: IF J>K T
HEN A$(H)=S$ :: GOTO 450
490 IF J>=K THEN 510
500 IF SEG$(A$(J),FS,FL)<SEG
$(A$(J+1),FS,FL)THEN J=J+1
510 IF SEG$(S$,FS,FL)>=SEG$(
A$(J),FS,FL)THEN A$(H)=S$ ::
 GOTO 450
520 A$(H)=A$(J):: GOTO 480
530 REM SAVE ON DISK (Y GOES
 TO X+2 SO LAST 2 LINES OF F
ILE ARE SAVED)
540 CALL CLEAR :: DISPLAY AT
(12,1):"Input device and fil
e name,":"   DSK";D$
550 ACCEPT AT(13,7)SIZE(-12)
:D$ :: IF D$="" THEN 240 ELS
E OPEN #1:"DSK"&D$,DISPLAY ,
VARIABLE 80,OUTPUT
560 FOR Y=0 TO X+2 :: PRINT
#1:A$(Y):: NEXT Y :: CLOSE #
1 :: GOTO 350

570 REM  QUIT
580 CALL CLEAR :: DISPLAY AT
(12,1):"QUIT- Are you sure?
Y/N"
590 ACCEPT AT(12,25)SIZE(1)V
ALIDATE("YN"):B$ :: IF B$="N
" THEN 350 ELSE END
600 PRINT "File not properly
 organized.": : :: GOTO 230
```



**** * EASY GRADER * ****

by Harold Hoyt

Now that my daughter, Kim, is a school teacher, I see that she can use all the help that she can get. I see her using a "slide rule table" for grading homework and tests. You move the number of problems on the test under a window and look up the percent right as a function of the number of questions missed. I thought that it might be handy to have several copies of this kind of table, produced by the computer for easy insertion in a notebook. Maybe all of the school teachers might find it handy.

In order to get a large table printed in a small space, a printer that can do condensed 136 characters and subscripts is required. The table covers a range of 4 to 99 problems. Some squirming was required to get everything to fit. The table is printed as three smaller tables. After each table is printed, the program stops to allow the operator to position the paper. After the paper is positioned as desired, press any key to continue. Do not turn off the printer to position the paper as the control codes to the printer are sent only once when the printer file is opened. The first two tables fit nicely on one 8.5 by 11" sheet and the third table nearly fills a second sheet.

One could make several copies of the table without separating the sheets and then put the paper back in the printer reversed so that the tables would print on both sides. One would have to stagger the printing by one sheet to come out even.

Problems 5 through 35 are in one table, 36 through 67 in a second, and 68 through 99 in the third. Line 100 opens the printer. Substitute codes as required for your printer int the string at the end of line 100. 27 65 06 sets line spacing at 6/72". 27 66 03 sets condensed. 27 92 01 slashes the zero and 27 83 01 sets subscripts. The 13 performs a carriage return to start a fresh line for the header.

For P=0 to 2 refers to pages or passes. Could have said T for tables? For C=4+32*P to 35+32*P allows the three tables to be non-overlapping. The rows are calculated to be one less than the maximum # of problems. The whole thing was designed without TAB settings using tricks to make each column entry right justified printing PRINT #1:RPT$(" ",3-LEN(C$))C$; :This function will use 3 printing spaces if C$ is 0,1,2 or 3 characters long.

The only meaningful calculation is in line 140 where C$=STR$(INT(100*((C-R)/C)+.5)). C is the total # of problems, R, the # wrong, is the row #. C-R/C is the fraction right. Multiplied by 100 is % right. Add 0.5 and do an INT rounds up to the nearest percent.

```
**************
* EASY GRADER *
*             *
* THE PROGRAM *
**************

1 !SAVE DSK1.GRADER !200
100 CALL CLEAR :: OPEN #1:"P
IO",VARIABLE 136 :: FOR C=1
TO 14 :: PRINT #1:CHR$(VAL(S
EG$("152765062766032792012 78
30113",2*C-1,2)));:: NEXT C
!147
110 X$="    Easy Grader
       by Harold Hoyt 10/1
1/88" :: DISPLAY AT(10,7):X$
 :: FOR P=0 TO 2 :: PRINT #1
:X$:TAB(60);"# of Problems"
!047
120 PRINT #1:" Wrong";:: FO
R C=4+32*P TO 35+32*P :: C$=
STR$(C):: PRINT #1:RPT$(" ",
3-LEN(C$))&C$;:: NEXT C :: P
RINT #1:" Wrong" !PrntHdr !1
43
130 FOR R=1 TO 34+P*32 :: R$
=STR$(R):: PRINT #1:TAB(8-LE
N(R$));R$;:: FOR C=4+32*P TO
 35+32*P :: C$="---" :: IF C
<R THEN 150 !173
140 C$=STR$(INT(100*((C-R)/C
)+.5))!238
150 PRINT #1:RPT$(" ",3-LEN(
C$))&C$;:: NEXT C :: PRINT #
1:RPT$(" ",3-LEN(R$))&R$ ::
NEXT R !135
160 DISPLAY AT(12,1):"" :: D
ISPLAY AT(12,1):"Press Any K
ey To Continue" :: CALL KEY(
0,K,S):: IF S=0 THEN 160 ::
DISPLAY AT(12,1):"Working" !
080
170 NEXT P :: CLOSE #1 !255
```

NOTCE !

EASY GRADER EXAMPLE

ON P. 8

15

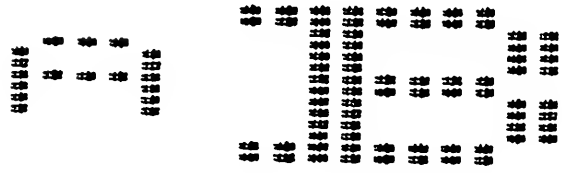
THE NEW CURSOR

++++++++++++++++

By Rich Renth

In this short little routine that follows, you can change your cursor to whatever character you please. The routine is an old but useful one. To change the character simply change the 8 numbers after the 12288 in line 110. These numbers must be in decimal form not hex. To convert your 16 character pattern indentifier (used in CALL ChAR) to decimal, split it up into 8 pairs for the 8 rows. The first number of the pair should be multiplied by 16 and added to the second. The letters in hex to decimal are A=10 B+11 C=12 D=13 E=14 F=15. Here is a small example of a pair -E7 = 14 X 16 + 7 or just 231. When this program is run, your new cursor will stay this way until you exit extended basic. Cursor below is small underline. HAVE FUN !!!!!

```
100 CALL INIT :: CALL LOAD(8
196,63,248):: CALL LOAD(1637
6,67,72,65,78,71,69,48,8
)
110 CALL LOAD(12288,0,0,0,0,
0,0,0,124)
120 CALL LOAD(12296,2,0,3,24
0,2,1,48,0,2,2,0,8,4,32,32,3
6,4,91):: CALL LINK("CHA
NGE")
```



BASIC BANNERS

by Steve Karasek

This short Extended Basic program will print banners. It asks you for a size from 1 to 10, and for a message to be printed. The message can be of any length. It will then print the message sideways on the printer in block letters. Two one-character examples are shown of size 1 and size 2. Size 10 is about the size of this page. You can mix sizes or create very long banners by running the program several times. If your printer is not addressed as "PIO", change the name in line 100.

The numbers at the end of each line are checksums. Use the CHECKSUM program by Tom Freeman (this months disk of the month) to enter the program.

```
100 OPEN #1:"PIO",OUTPUT !16
7
110 PRINT "SIZE (1-10)";:: I
NPUT SZ :: PRINT "ENTER MESS
AGE:" :: LINPUT X$ :: B$=RPT
$(" ",SZ)!002
120 FOR I=1 TO LEN(X$):: Z$=
SEG$(X$,I,1):: CALL CHARPAT(
ASC(Z$),PAT$):: D$=RPT$(Z$,S
Z):: FOR C=0 TO 5 :: O$(C)="
" :: NEXT C !021
130 FOR A=15 TO 3 STEP -2 ::
 N=POS("0123456789ABCDEF",SE
G$(PAT$,A,1),1)-1 !047
140 FOR C=0 TO 3 :: IF 2^(3-
C)AND N THEN O$(C)=O$(C)&D$
ELSE O$(C)=O$(C)&B$ !231
150 NEXT C :: N=POS("0123456
789ABCDEF",SEG$(PAT$,A+1,1),
1)-1 :: FOR C=4 TO 5 :: IF 2
^(7-C)AND N THEN O$(C)=O$(C)
&D$ ELSE O$(C)=O$(C)&B$ !095
160 NEXT C :: NEXT A :: FOR
C=0 TO 5 :: FOR J=1 TO SZ ::
 PRINT #1:TAB(41-LEN(O$(C))/
2);O$(C):: NEXT J :: NEXT C
:: NEXT I !148
```

CLOCKS REVISITED

by Harold Hoyt

The program listing from our Nov 88 newsletter 'TICKS' won't work from a cold start without adding a line, 90 CALL PEEK(8198,A) :: if A<>170 THEN CALL INIT, which is required for the program to RUN if INIT hasn't been called.

The program, as originally written, converts HEX DATA to base 10 data and uses "CALL LOAD" in XBasic to put the converted DATA into memory. You may substitute any HEX DATA strings representing a MEMORY IMAGE program that is RUNnable from XBasic for the DATA statements in this program, which makes it a dandy utility.

I have chosen to rework Mr. Fitchhorn's program, so that it will write the "CALL LOAD" stuff to a TEXT file, which can then be converted to a program, which may be SAVed and RUN. The extra steps required to create this new program are worth while, since you only do the extra steps once to create the new program, and the resulting new program will run much faster.

Now for the details: Two approaches have been traditionally used for having programs write other programs. One opens a file in merge format, D/V 163, and sends Basic line number DATA and Tokens to this file. The end program is then recovered by using a MERGE format to convert the D/V 163 file to an XBasic program. The second approach opens a D/V 80 file and writes a program listing to the file. The listing is then converted to an XBasic program using a utility that converts a listing to a program, such as "ENTER" in SUPERBASIC, using a TOKEN look-up table. I like the second approach better since it allows one to convert any TEXT file program listing from any machine to TI XBasic.

Assume that you have some source code that could be assembled and RUN using XBasic, such as Mr. Fitchhorn's clock program. Edit out of Mr. Fitchhorn's program just about everything but the DATA. Then, attach a subroutine that will write one CALL LOAD line of text, including the line number. Lines 1500 to 1600 do this, with opening an output file at line 110. Line 120 sets up the line # for the listing, the decimal poke address and the H$ used in the HEX to DEC conversion. Lines 130,140 demonstrate the ability to POKE address lines to the listing directly. Lines 500 to 990 POKE the last lines to the listing, including the look-up of the "LFAL" etc. "last free address in low memory" etc.

If you have the source code of a program that will run in XBasic and want to convert it to CALL LOAD format, the following is one way of doing it, using TICK/HH as a UTILITY. To generate the HEX DATA for the program, use the Editor/Assembler cartridge and E/A disk, (not Funnelweb) to edit the source code and place an AORG statement before the first executable statement. Ordinarily, the assembler will put the first batch of relocatable code at this address. We will take over the responsibility of placing the code properly with our CALL LOAD statements. We need to update the FFAM (First Free Address In Memory). The reason I want to use the E/A cartridge version of the ASSEMBLER is to create a LISTING that shows the Source and object code. The AORG statement forces the assembler listing to show non-relocatable code in the LISTing. This only done for you if you load the program from Editor/Assemmbler. Be sure and NOT use the "C" Compressed object code option as this is not useable from XBasic

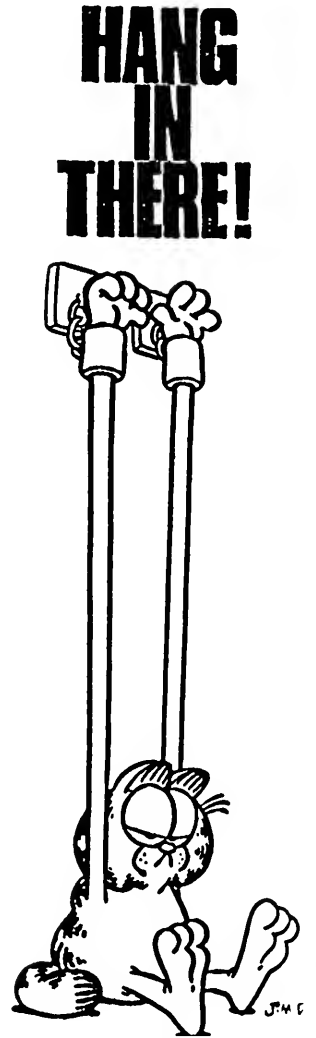
Well, I've finally caught up with what Dr. Tamashiro was doing 2 years ago. Don't give up on assembly language! It's the way to go!

TICK/HH

```
1 !SAVE DSK1.TICK/HH !253
10 !----------------------!
 !175
20 ! EXTENDED             !
 !213
30 ! BASIC program to load !
 !033
40 ! Interrupt driven clock!
 !101
50 !  By: D. L. Fitchhorn  !
 !010
60 !      305 Navajo       !
 !091
70 !      Keller, TX  76248!
 !006
80 !----------------------!
 !175
90 !Modified by H. Hoyt to w
rite prog listing 1/3/89 !14
0
100 !Writes data to a CALL L
OAD program listing. You con
vert listing to XBasic and t
hen RUN it. !114
110 OPEN #1:"DSK1.LISTING",O
UTPUT !035
120 LN=500 :: R=10240 :: T=2
4 :: HH,MN=0 :: HX$="0123456
789ABCDEF" !101
130 PRINT #1:"1 !SAVE DSK1.H
CLOCK" !Let's Keep track of
what program is where! !045
140 PRINT #1:STR$(LN);" CALL
 PEEK(8198,A):: IF A()170 TH
EN CALL INIT" :: LN=LN+1
0 !192
150 GOSUB 1500 !049
160 IF X$()"END" THEN 150 !0
47
500 !EXIT Routine !195
510 LN=LN+10 :: X$=STR$(LN)&
" CALL PEEK(8196,A,B) :: LFA
L=A*256+B  :: NEWL=LFAL-
16 :: A=INT(NEWL/256) :: B=N
EWL-A*256" !101
515 PRINT #1:X$ !196
520 LN=LN+10 :: PRINT #1:STR
$(LN);" CALL LOAD(8196,A,B):
: CALL LOAD(NEWL,83,84,65,82
,84,32,40,0)" !170
530 LN=LN+10 :: PRINT #1:STR
$(LN);" CALL LOAD(NEWL+8,83,
84,79,80,32,32,40,40):: CALL
 LINK(""START"")" !108
540 LN=LN+10 :: PRINT #1:STR
$(LN);" CALL LOAD(10543,T,0,
HR,0,MN,0,SC)" !212
990 CLOSE #1 :: END !164
1000 DATA C820,28EA,292C,C82
0,28E8,2928,C820,28E6 !224
1010 DATA 292E,04E0,2930,04E
0,2932,04E0,2934,0200 !119
1020 DATA 282E,C800,83C4,045
B,04E0,83C4,045B !104
1030 DATA 02E0,2928,02E0,292
8,0602,1652,C0A0,28EA !154
1040 DATA 0586,0286,003C,160
E,04C6,0585,0285,003C !129
1050 DATA 1609,04C5,0584,80C
4,1605,04C4,0283,0018,1301,0
584 !005
1060 DATA 06C0,D800,8C02,06C
0,E020,28E4,D800,8C02 !182
1070 DATA 4020,28E4,D064,28E
C,0941,0221,9000,D801,8C00 !
228
1080 DATA 0A41,0241,0F00,022
1,9000,D801,8C00,0201 !084
1090 DATA 9A00,D801,8C00,D06
5,28EC,0941,0221,9000 !151
1100 DATA D801,8C00,0A41,024
1,0F00,0221,9000,D801 !110
1110 DATA 8C00,0201,8C00,D80
1,8C00,D066,28EC,0941 !169
1120 DATA 0221,9000,D801,8C0
0,0A41,0241,0F00,0221 !086
1130 DATA 9000,D801,8C00,072
0,8306 !138
1140 DATA 02E0,83E0,045B,400
0,000D,0017,003B !032
1150 DATA 0001,0203,0405,060
7,0809,1011 !138
1160 DATA 1213,1415,1617,181
9,2021,2223 !156
1170 DATA 2425,2627,2829,303
1,3233,3435 !174
1180 DATA 3637,3839,4041,424
3,4445,4647 !192
1190 DATA 4849,5051,5253,545
5,5657,5859,END !039
1500 !SUB WRITE LINE(LN,R,X$
,A,B) !076
1510 READ X$ !019
1520 IF X$="END" THEN 1600 !
030
1530 PRINT #1:STR$(LN);" CAL
L LOAD(";STR$(R);",";!254
1540 FOR I=1 TO 8 !063
1550 A=(POS(HX$,SEG$(X$,1,1)
,1)-1)*16+POS(HX$,SEG$(X$,2,
1),1)-1 :: B=(POS(HX$,SEG$(X
$,3,1),1)-1)*16+POS(HX$,SEG$
(X$,4,1),1)-1 !207
1560 PRINT #1:STR$(A);",";ST
R$(B);:: R=R+2 :: IF I=8 THE
N 1590 !206
1570 READ X$ :: IF X$="END"
THEN 1590 !169
1580 PRINT #1:",";:: NEXT I
!081
1590 PRINT #1:")" :: LN=LN+1
0 !153
1600 RETURN !136
```

HCLOCK

```
1 !SAVE DSK1.HCLOCK !199
500 CALL PEEK(8198,A):: IF A
()170 THEN CALL INIT !011
510 CALL LOAD(10240,200,32,4
0,234,41,44,200,32,40,232,41
,40,200,32,40,230)!252
520 CALL LOAD(10256,41,46,4,
224,41,48,4,224,41,50,4,224,
41,52,2,0)!139
530 CALL LOAD(10272,40,46,20
0,0,131,196,4,91,4,224,131,1
96,4,91,2,224)!093
540 CALL LOAD(10288,41,40,2,
224,41,40,6,2,22,82,192,160,
40,234,5,134)!038
550 CALL LOAD(10304,2,134,0,
60,22,14,4,198,5,133,2,133,0
,60,22,9)!079
560 CALL LOAD(10320,4,197,5,
132,128,196,22,5,4,196,2,131
,0,24,19,1)!200
570 CALL LOAD(10336,5,132,6,
192,216,0,140,2,6,192,224,32
,40,228,216,0)!087
580 CALL LOAD(10352,140,2,64
,32,40,228,208,100,40,236,9,
65,2,33,144,0)!085
590 CALL LOAD(10368,216,1,14
0,0,10,65,2,65,15,0,2,33,144
,0,216,1)!075
600 CALL LOAD(10384,140,0,2,
1,154,0,216,1,140,0,208,101,
40,236,9,65)!227
610 CALL LOAD(10400,2,33,144
,0,216,1,140,0,10,65,2,65,15
,0,2,33)!011
620 CALL LOAD(10416,144,0,21
6,1,140,0,2,1,140,0,216,1,14
0,0,208,102)!203
630 CALL LOAD(10432,40,236,9
,65,2,33,144,0,216,1,140,0,1
0,65,2,65)!135
640 CALL LOAD(10448,15,0,2,3
3,144,0,216,1,140,0,7,32,131
,214,2,224)!171
650 CALL LOAD(10464,131,224,
4,91,64,0,0,13,0,23,0,59,0,1
,2,3)!125
660 CALL LOAD(10480,4,5,6,7,
8,9,16,17,18,19,20,21,22,23,
24,25)!206
670 CALL LOAD(10496,32,33,34
,35,36,37,38,39,40,41,48,49,
50,51,52,53)!023
680 CALL LOAD(10512,54,55,56
,57,64,65,66,67,68,69,70,71,
72,73,80,81)!052
690 CALL LOAD(10528,82,83,84
,85,86,87,88,89)!023
710 CALL PEEK(8196,A,B):: LF
AL=A*256+B :: NEWL=LFAL-16 :
: A=INT(NEWL/256):: B=NEWL-A
*256 !227
720 CALL LOAD(8196,A,B):: CA
LL LOAD(NEWL,83,84,65,82,84,
32,40,0)!041
730 CALL LOAD(NEWL+8,83,84,7
9,80,32,32,40,40):: CALL LIN
K("START")!247
740 CALL LOAD(10543,T,0,HR,0
,MN,0,SC)!175
```



```
**********************
*R E S I S T O R S   *
*         by         *
*   H. C. Hoyt jr.   *
**********************
```

I'm thouroughly convinced that computers are the perfect teaching tool. All that is needed is good software. To prove my point, this month, I am presenting one of my vintage programs, written to run in either Basic or XBasic. It is a training aid for people who want to learn the resistor color code universally used by electronics people. The colors black, brown, red, orange, yellow, green, blue, violet, grey and white represent the digits 0 through 9 respectively. In the program, a resistor is displayed. When the program is first run, you are supposed to read the colors from the resistor and figure it's resistance. press any key but quit or <ENTER> and the answer is written above the resistor. Press <ENTER> and the order is reversed. The value is written above the resistor and upon pressing a key, the colors appear. Do this long enough, and you will not only learn how to read resistor color codes, but also what values of resistors are normally used. Since not everyone is using a color monitor and NTSC colors are poor anyways, the colors are written out on the screen.

This program is superior to another that I've seen that purports to teach the color code, since it uses a look up list of real resistors in a table, rather than totally randomly generating unreal resistors. No one really uses a 723 ohm resistor. The values of resistance this program displays are all standard stock resistors. The math buffs in the crowd might enjoy figuring out how the 3rd digit was generated.

```
1 REM SAVE DSK1.RESISTOR* !2
39
100 REM ****************** !
174
110 REM *  (RESISTORS)   * !
246
120 REM * BY HAROLD HOYT * !
039
130 REM * TRAINING AID   * !
216
140 REM  * 2/11/85       * !
141
150 REM ****************** !
174
160 DIM R(160)!171
170 CALL SCREEN(3)!148
180 GOSUB 9000 !155
190 GOSUB 1000 !059
200 IF FLAG()1 THEN 330 !219
210 ROW=4 !179
220 COL=3 !152
230 ZZ1$="?:COLORS,ANSWER=VA
LUE" !167
240 GOSUB 6000 !215
250 GOSUB 5000 !235
260 REM CLEAR OLD ANSWER (VA
LUE) !039
270 CALL HCHAR(3,3,32,20)!16
9
280 CALL HCHAR(3,3,32,20)!16
9
290 GOSUB 3000 !019
300 GOSUB 8000 !175
310 GOSUB 7000 !195
320 GOSUB 8000 !175
330 IF FLAG=1 THEN 200 !151
340 ROW=4 !179
350 COL=3 !152
360 ZZ1$="?:VALUE,ANSWER=COL
ORS" !167
370 GOSUB 6000 !215
380 GOSUB 6000 !215
390 GOSUB 5000 !235
400 GOSUB 7000 !195
410 REM CLEAR OLD ANSWER (BA
NDS & VERTICAL LABELS) !165
420 CALL COLOR(10,1,1)!219
430 CALL COLOR(12,1,1)!221
440 CALL COLOR(13,1,1)!222
450 CALL COLOR(14,1,1)!223
460 CALL VCHAR(14,10,32,6)!2
36
470 CALL VCHAR(14,13,32,6)!2
39
480 CALL VCHAR(14,16,32,6)!2
42
490 CALL VCHAR(14,19,32,6)!2
45
500 GOSUB 8000 !175
510 GOSUB 3000 !019
520 GOSUB 8000 !175
530 GOTO 200 !023
990 STOP !152
1000 REM ****************** 18
!174
1010 REM * INITIALIZATION *
!118
1020 REM * LOAD PREFERRED *
!045
1030 REM * VALUES OF      *
!115
1040 REM * RESISTORS      *
!188
1050 REM ****************** 
!174
1070 FLAG=1 !210
1080 FOR I=1 TO 20 !106
1090 READ R(I)!159
1100 R(I+20)=10*R(I)!169
1110 R(I+40)=100*R(I)!220
1120 R(I+60)=1000*R(I)!015
1130 R(I+80)=1E4*R(I)!249
1140 R(I+100)=1E5*R(I)!036
1150 R(I+120)=1E6*R(I)!039
1160 R(I+140)=1E7*R(I)!042
1170 REM USE VALUES TO 22 ME
GOHM=R(148) !056
1180 NEXT I !223
1190 DATA 1,1.1,1.2,1.4,1.5
!205
1200 DATA 1.8,2,2.2,2.4,2.7
!218
1210 DATA 3,3.3,3.6,3.9,4.7
!229
1220 DATA 5.1,6.8,7.5,8.2,9.
1 !080
1230 FOR I=1 TO 3 !058
1240 READ TOL$(I),TOL(I),TOL
COL(I)!238
1250 NEXT I !223
1260 DATA "GOLD  ",5,12,"SIL
VER",10,15,"      ",20,1 !05
7
1500 REM CONSTRUCT RESISTOR
!026
1510 REM DEFINE COLOR BAND G
ROUPS !057
1520 RESTORE 1560 !123
1530 FOR I=0 TO 9 !063
1540 READ COLOR$(I),COLR(I)!
137
1550 NEXT I !223
1560 DATA "BLACK ",2,"BROWN
",11,"RED   ",7,"ORANGE",9,"
YELLOW",12 !045
1570 DATA "GREEN ",13,"BLUE
 ",8,"PURPLE",14,"GREY  ",15
,"WHITE ",16 !135
1580 REM INITIAL BANDS TRANS
PARENT !190
1590 CALL COLOR(10,1,1)!219
1600 CALL COLOR(12,1,1)!221
1610 CALL COLOR(13,1,1)!222
1620 CALL COLOR(14,1,1)!223
1630 REM DRAW RESISTOR !131
```

```
1640 CALL CLEAR !209
1650 CALL CHAR(112,"000000FF
FF000000")!053
1660 CALL CHAR(113,"COCOCOCO
COCOCOCO")!118
1670 CALL CHAR(114,"00000000
0000FFFF")!055
1680 CALL CHAR(115,"FFFF0000
00000000")!056
1690 CALL CHAR(116,"03030303
03030303")!249
1700 CALL CHAR(117,"183C7EFF
1818181818")!109
1710 CALL HCHAR(8,3,112,5)!1
76
1720 CALL VCHAR(6,8,113,5)!1
94
1730 CALL HCHAR(5,8,114,17)!
232
1740 CALL HCHAR(11,8,115,17)
!023
1750 CALL VCHAR(6,24,116,5)!
244
1760 CALL HCHAR(8,25,112,4)!
228
1770 CALL CHAR(120,"FFFFFFFF
FFFFFFFF")!060
1780 CALL CHAR(128,"FFFFFFFF
FFFFFFFF")!068
1790 CALL CHAR(136,"FFFFFFFF
FFFFFFFF")!067
1800 CALL CHAR(104,"FFFFFFFF
FFFFFFFF")!062
1810 CALL VCHAR(6,10,120,5)!
234
1820 CALL VCHAR(6,13,128,5)!
245
1830 CALL VCHAR(6,16,136,5)!
247
1840 CALL VCHAR(6,19,104,5)!
245
1850 CALL VCHAR(12,10,117)!1
09
1860 CALL VCHAR(12,13,117)!1
12
1870 CALL VCHAR(12,16,117)!1
15
1880 CALL VCHAR(12,19,117)!1
18
1890 ZZ1$="PRESS ANY KEY TO
CONTINUE" !141
1900 ROW=20 !226
1910 COL=4 !153
1920 GOSUB 6000 !215
1930 ZZ1$="PRESS (ENTER) TO
CHANGE" !211
1940 ROW=21 !227
1950 GOSUB 6000 !215
1960 ZZ1$="ORDER OF DISPLAY"
 !005
1970 ROW=22 !228
1980 GOSUB 6000 !215
1990 ZZ1$="PRESS (S) TO EXIT
" !054
2000 ROW=23 !229
2010 GOSUB 6000 !215
2020 RETURN !136
3000 REM DEFINE BANDS & LABE
LS !006
3010 REM COLOR BANDS !193
3020 CALL COLOR(10,TOLCOL(DI
GIT4),8)!199
3030 CALL COLOR(12,COLR(DIGI
T1),8)!041
3040 CALL COLOR(13,COLR(DIGI
T2),8)!043
3050 IF DIGIT3<0 THEN 3080 !
109
3060 CALL COLOR(14,COLR(DIGI
T3),8)!045
3070 GOTO 3090 !109
3080 CALL COLOR(14,11,8)!024
3090 REM GET RESISTOR LABELS
 !008
3100 LABEL1$=COLOR$(DIGIT1)!
037
3110 LABEL2$=COLOR$(DIGIT2)!
039
3120 IF DIGIT3<0 THEN 3150 !
179
3130 LABEL3$=COLOR$(DIGIT3)!
041
3140 GOTO 3160 !179
3150 LABEL3$="GOLD  " !168
3160 LABEL4$=TOL$(DIGIT4)!15
5
4000 REM PRINTS COLOR LABELS
 !012
4010 FOR LL=1 TO LEN(LABEL1$
)!132
4020 LL1$=SEG$(LABEL1$,LL,1)
!157
4030 CALL HCHAR(13+LL,10,ASC
(LL1$))!139
4040 NEXT LL !046
4050 FOR LL=1 TO LEN(LABEL2$
)!133
4060 LL2$=SEG$(LABEL2$,LL,1)
!159
4070 CALL HCHAR(13+LL,13,ASC
(LL2$))!143
4080 NEXT LL !046
4090 FOR LL=1 TO LEN(LABEL3$
)!134
4100 LL3$=SEG$(LABEL3$,LL,1)
!161
4110 CALL HCHAR(13+LL,16,ASC
(LL3$))!147
4120 NEXT LL !046
4130 FOR LL=1 TO LEN(LABEL4$
)!135
4140 LL4$=SEG$(LABEL4$,LL,1)
!163
4150 CALL HCHAR(13+LL,19,ASC
(LL4$))!151
4160 NEXT LL !046
4170 RETURN !136
5000 REM RANDOM SELECT RESIS
TOR !246
5010 RANDOMIZE !149
5020 RR=R(INT(148*RND+1))!02
6
5030 RR10=10*RR !085
5040 DIGIT3=INT(LOG(RR)/LOG(
10))-1 !103
5050 DIGIT1=VAL(SEG$(STR$(RR
),1,1))!050
5060 DIGIT2=VAL(SEG$(STR$(RR
10),2,1))!149
5070 REM DIGIT 4=TOLERANCE !
089
5080 REM 1=GOLD,5% 2=SILVER
!056
5090 REM 10% 3=NOSTRIPE,20%
!247
5100 DIGIT4=INT(2.2*RND)+1 !
081
5110 RETURN !136
6000 REM HORIZONTAL DISPLAY
RTN !014
6010 FOR LL=1 TO LEN(ZZ1$)!2
16
6020 LL1$=SEG$(ZZ1$,LL,1)!24
1
6030 CALL HCHAR(ROW,COL+LL-1
,ASC(LL1$))!196
6040 NEXT LL !046
6050 RETURN !136
7000 REM VALUE DISPLAY RTN !
129
7010 REM CLEAR PREVIOUS !190
7020 REM DISPLAY !208
7030 CALL HCHAR(3,3,32,20)!1
69
7040 REM MAKE + OR - SYMBOL
!039
7050 REM OUT OF "c" !046
7060 CALL CHAR(99,"1818FFFF1
818FFFF")!142
7070 ON DIGIT3+2 GOTO 7080,7
100,7120,7140,7160,7180,7200
,7220 !249
7080 ZZ1$=STR$(DIGIT1)&"."&S
TR$(DIGIT2)&" OHMS c "&STR$(
TOL(DIGIT4))&" %" !074
7090 GOTO 7230 !169
7100 ZZ1$=STR$(DIGIT1)&STR$(
DIGIT2)&" OHMS c "&STR$(TOL(
DIGIT4))&" %" !156
7110 GOTO 7230 !169
7120 ZZ1$=STR$(DIGIT1)&STR$(
DIGIT2)&"0 OHMS c "&STR$(TOL
(DIGIT4))&" %" !205
7130 GOTO 7230 !169
7140 ZZ1$=STR$(DIGIT1)&"."&S
TR$(DIGIT2)&" KOHMS c "&STR$
(TOL(DIGIT4))&" %" !150
7150 GOTO 7230 !169
7160 ZZ1$=STR$(DIGIT1)&STR$(
DIGIT2)&" KOHMS c "&STR$(TOL
(DIGIT4))&" %" !232
7170 GOTO 7230 !169
7180 ZZ1$=STR$(DIGIT1)&STR$(
DIGIT2)&"0 KOHMS c "&STR$(TO
L(DIGIT4))&" %" !025
7190 GOTO 7230 !169
7200 ZZ1$=STR$(DIGIT1)&"."&S
TR$(DIGIT2)&" MEGOHMS c "&ST
R$(TOL(DIGIT4))&" %" !038
7210 GOTO 7230 !169
7220 ZZ1$=STR$(DIGIT1)&STR$(
DIGIT2)&" MEGOHMS c "&STR$(T
OL(DIGIT4))&" %" !120
7230 ROW=3 !178
7240 COL=3 !152
7250 GOSUB 6000 !215
7260 RETURN !136
8000 REM KEY ENTRY RTN !105
8010 CALL SOUND(100,440,0)!1
26
8020 CALL KEY(0,K,S)!187
8030 IF S=0 THEN 8020 !122
8040 CALL SOUND(100,880,0)!1
34
8050 IF K=ASC("S")THEN 990 !
075
8060 REM TEST FOR "(ENTER)"
!093
8070 IF K<>13 THEN 8110 !194
8080 FLAG=-FLAG !180
8090 CALL SOUND(200,440,0,55
0,0)!067
8100 REM REVERSE ORDER !114
8110 RETURN !136
9000 REM ENTRY SCREEN DISPLA
Y  !194
9010 CALL CLEAR !209
9020 CALL CHAR(91,"182442810
0000000")!208
9030 CALL CHAR(92,"000000008
1422418")!209
9040 FOR COL=1 TO 31 STEP 2
!174
9050 CALL VCHAR(1,COL,91,24)
!160
9060 CALL VCHAR(1,COL+1,92,2
4)!092
9070 NEXT COL !116
9080 ROW=8 !183
9090 COL=4 !153
9100 ZZ1$="***R E S I S T O
R S***" !111
9110 GOSUB 6000 !215
9120 ROW=12 !227
9130 ZZ1$="   BY HAROLD HOY
T   " !125
9140 GOSUB 6000 !215
9150 ROW=16 !231
9160 ZZ1$="   FEBRUARY,1985
   " !039
9170 GOSUB 6000 !215
9180 RETURN !136
```

HARDWARE
and
EVERYTHING ELSE

GENEALOGY AND THE DATA BASE

I recently had the opportunity to meet several genealogist that were TI-99/4A owners. As you would expect, our conversation soon became that of research, and how to get the computer to do some of the tedious work that needs to be done. This would leave us more time for the actual research. To our dismay no one knew of a TI program that would do what we wanted. I then told them what I have done to make my research easier. I am now telling you.

Where most of the Genealogy programs fail is in the data base. They use the family groupsheet as the template for the data base. I conclude that this is too little too late. By the time the genealogist has made it this far he no longer needs a data base, except for indexing. Where the data base is needed is when sorting and searching through vital records for common denominators. Many families have the same name passed on generation after generation. It often becomes difficult to decipher one from the other but could be calculated from comparing vital records. The Mormons understand this therefore, the Latterday Saints CFI data base. Unfortunately, this program is not as yet available for the TI-99/4A.

It has long been my opinion that no ONE program would do all that I wanted to do with my genealogy, but that TWO would. What you really need is a good word processor (in our case TI-WRITER our one of the TI-WRITER clones) and a good data base. The problem then becomes what is a good data base for the TI? Good is a relative term. What is good for some things does not always apply to something else. Here is what I used as my criteria:

#1 EASE OF USE - Any program that I use must be user friendly. I do not want to waste precious time playing with, or wading through massive documentation to make it do what I want. Better yet, it should do all that it claims. There is nothing worse than transvering all your data to then find out it will not do what was expected. It needs to be a time-saving device, not a time-making device. The Genealogist would rather be spending his time doing research. If it does not save him time, he will not waste the time to use it.

#2 FILE STRUCTURE - The program should be able to search and sort all fields. A subsort would be nice. That way when a surname is duplicated it would go to the next field and alaphabetized by that field, and so on, much the way we were taught in our childhood. If a subsort is not posssible field length should be a minimum of 28 characters. This would allow last/first name alphabetization. PROGRAM SHOULD BE ABLE TO ACCEPT MANY FIELDS (approx. 15).

#3 PRINT FILE - The program should print in PAGE AND REPORT format. Report format makes for easy reference and makes duplication more obvious. Page format for those who like to enter their data on index cards. PRINTER DEVICE NAME should be available from a title screen and not embedded in the program! If this is not possible line #'s for change should be highlighted in the text for possible change. There is nothing more frustrating then sending a file to the printer and have it just sit there doing nothing because there is a hand-shaking problem in the program.

#4 SAVE FILE - Of course this is a must for any data base. It would be nice, however, if the files could be saved to work with other software. This is not a must, but could be helpful in adding notes or comments to file.

SUGGESTED FILE STRUCTURE

FIELD 1 -	SURNAME
FIELD 2 -	FIRSTNAME
FIELD 3 -	FATHER
FIELD 4 -	MOTHER
FIELD 5 -	SPOUSE
FIELD 6 -	SEX
FIELD 7 -	TYPE (of records)
FIELD 8 -	EVENT DATE (*)
FIELD 9 -	EVENT PLACE
FIELD 10-	SOURCE (of reference *)
FIELD 11-	OPEN FIELD (*)

FAMILY RECORD FILE USING P.R.K. MODULE
by Jan Knapp

Here's a little something for the Genealogical Buff or just keeping the family records. The file structure is one similar to that used at the LDS library system, based in Salt Lake City. I have found the P.R.K. search capabilities a real pain saver, since it will search and sort in all 12 fields.

Like all programs it has it's limitations. You will need to set up your own abreviations for the type of record found. My personal choice is: B=BIRTH; C=CHRISTENED; D=DEATH; M=MARRIAGE; N=CENSUS; BI=BIOGRAPHY; F=FAMILY RECORDS; A=ADOPTION; DV=DIVORCE; ML=MILITARY; BR=BURIED; CM=CEMETARY. This program is limited in size I suggest that several files should be made, and kept under the base surname of that file, if your records exceed pages allowed.

I have entered two extra field for possible footnotes, additional marriages, etc. Although, I personally use this program for Genealogy it can be used to keep family events and members current.

FILE STRUCTURE

	ITEM	TYPE	WIDTH	DEC
1	SURNAME	CHAR	15	0
2	F/NAME	CHAR	15	0
3	SEX	CHAR	2	0
4	TYPE	CHAR	2	0
5	FATHER	CHAR	15	0
6	MOTHER	CHAR	15	0
7	SPOUSE	CHAR	15	0
8	E/DATE	CHAR	10	0
9	E/PLACE	CHAR	15	0
10	SOURCE	CHAR	15	0
11	MISC.	CHAR	15	0

TEMPLATES
by Jan Knapp

OR...HOW TO MAKE YOUR OWN FILL IN THE BLANK. Have you ever found yourself typing the same material over and over again and wished that you had a form that all you had to do was load it up and fill in the blanks? Well you can. I came across this problem with my Genealogy. It seemed that all I ever did was type. If I wanted to send a copy of my groupsheet, or change some information on it I had to type it all over. I had to use a regular typewriter or hand write the material. This constant duplication got me down. I decided that enough was enough. You see I'm basically lazy and will find any short cut I can. So I made my own Family Group Sheet on the TI-WRITER , and a whole new world opened up! I can now load my file, fill in the blanks, change material, and save it again for a later date if necessary. Now when I send my family sheet to someone all I do is load up the program and let the printer do the rest. Sound like something you could use? Then keep reading.

Now comes the hard part. Why is it that the simplest things to do are the hardest to explain? Well here goes. Your TI-WRITER, if you're using standard paper and normal print allows you 80 characters, or spaces, across and about 66 down. The default on TI-WRITER is set on 66. For your purposes I suggest less than this. There should be some spacing at the top and bottom for appearence, 64 is a better number to work with. Decide what material you wish to FORM, maybe you already have a form that you are using. If not imagine a grid 80 spaces across and 64 lines down (graph paper works great). This will allow you 5,120 posssible chracters to work with, unless you set margins. The worst part is figuring the proper amount of spaces for the blanks. Proper spacing is the hardest part of the whole process. REMEMBER!!! When working with TEMPLATES leave the WORD WRAP(the solid cursor) OFF! Word Wrap has some built-ins that you don't want. Now you have your hollow cursor ON, your tabs set to your needs, and are ready to go. After you have all the needed fill in materail done, and you have some left over space, you can leave the rest for NOTES: or COMMENTS:. By putting this material at the bottom it will allow you to turn on your word wrap for lengthly remarks (much easier on yourself).

Now you have your TEMPLATE made to your satisfaction, SAVE IT. If you are using DISK remember never to duplicate names, as you will erase a file. Give your Template a name of it's own, the files another. Now type in your information. CAUTION!!! Avoid insert, delete, and other TI-WRITER commands if you can, as these could possibly reformat your TEMPLATE beyond recognition. If you do wish to insert and delete remember that they must balance. It may seem like a lot of work, but remember you only have to do it ONCE. After you've finished your TEMPLATE save it if you wish (under a new name). Reload your TEMPLATE and you're ready for the next. This is much easier than it appears.

XBASIC PROGRAM TIP
by Harold Hoyt

Here is a handy little XBASIC programming tip. Lots of programmers put their program identification in line 100 as a REMark. Six Months later, listing the program, it answers your questions. What the heck is this? When did I write it? Why did I write it?

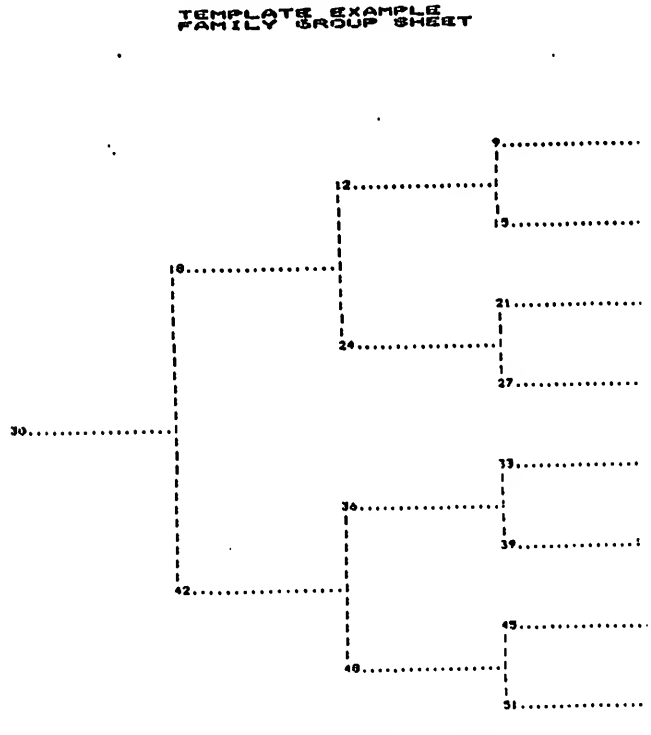
Recently, I've been pulling a little trick in XBASIC that the more I do it, the handier it seems. I add another program line, so that the beginning of my program looks like this:

```
1 !SAVE DSK1.PROGRAMNAM
100 !Prog 'PROGRAMNAM' H Hoyt 7/28/87 Demonstration Prog
```

Note all of the 10 characters in the program name can be used to truly describe the program. One of my friends was using single letter program names because he hated the extra typing.

If you type line 100 program description data once, and then hit <ENTER> followed by <FCTN 8> (REDO), you can edit out 00 in 100 to create line 1 and further edit it to create SAVE DSK1.PROGRAMNAM and delete the rest of the stuff. By itself, this wouldn't be worth the trouble, but just see what happens next.

After line 1 has been properly <ENTERED>, hit <FCTN REDO> again and then <FCTN DELETE> to eat the line #, space and !. What remains is SAVE DSK1.PROGRAMNAM in the screen buffer. Just hit <ENTER> and the program is automatically saved. This saves so much typing that it encourages you to save program pieces more frequently, reducing the loss in case of a program crash.



MARGINS SET AT 0 AND 79. TABS SET AT LOCATIONS 20,40, AND 60. LINE NUMBERS ARE LISTED ON GRAPH.

23

****
* PRK FILE STRUCTURE *
* for '87 INDEX *
****

FILE STRUCTURE

NAME: 87INDEX

DATE: 1/8/88

ITEMS/PAGE: 6

PAGES USED: 77

PAGES LEFT: 58

FILE STRUCTURE

ITEM		TYPE	WIDTH	DEC
1	TITLE OF	CHAR	15	0
2	ARTICLE	CHAR	15	0
3	ISSUE	CHAR	4	0
4	AUTHOR	CHAR	15	0
5	TYPE	CHAR	10	0
6	REQUIRED	CHAR	10	0

The above is the PRK (Personal Record Keeping) file structure used to compile the 1987 Computer Bridge newsletters. It is very versatile and can search and sort in all 6 fields. I can now sort alphabetically by Article, Author, etc. Most importantly to me is that it will print out in report format. With the aid of PRG (Personal Report Generator) it is camera ready to go to the printers as is. It may not be as pretty as last years (done on TI-WRITER) but much easier. As you can tell by the file structure, there was more than enough memory to do the job.

FROM ALL OF US

TO ALL OF YOU

HAVE A HAPPY 1988!

THE FORMIDABLE FORMATER
by Jan Knapp

```
* TEXT FORMATTER * FIX1.

ENTER INPUT FILENAME:

DSK2.
```

Do the above lines frighten you? At one time they did me. I found out that they were not as formidable as I had once feared. I often wondered what would happen after I put my poor defenceless file in there. Would I ever recognize it again? What would it do to it. It must be something really awful since it wouldn't let me see it once it had it in it's grasp. A member of our club gave me no choice one day and I have been greatful ever since.

Before this day my only contact with the formatter had been in printing out articles that had been sent to us in that form. Ours was a distant relationship. I was just a go between, a body being used to communicate someone elses wishes to the machine. I was not responsible if it didn't turn out right (it always did) since I hadn't written it, someone else had. It wasn't MY problem. This time it was different. We couldn't run his article as it was. We needed to change it's size to fit the space allowed in the newsletter and we really wanted to run the article. There was no choice except a do-it-yourself crash course in formatter. We printed the article out on EDITOR and disected it from there. About 30 minutes later we had it OUR way, and I wondered why I had been so afraid to try before. From then on I became braver and braver, to now when I'm attempting to teach TI-WRITER courses. Thank you, Steve!

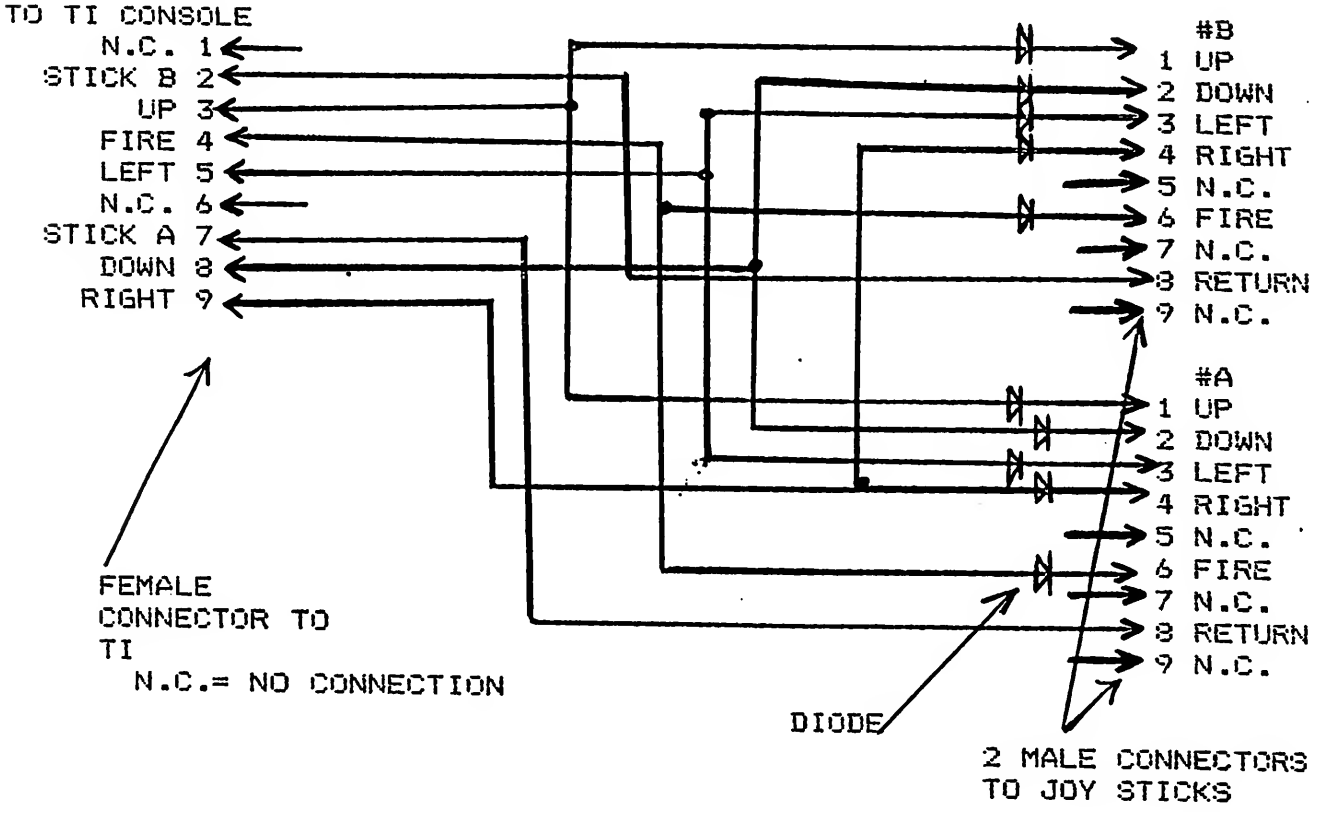
As far as my original fears, they were unfounded. The FORMATTER is actually rather niave. When it takes your file it doesn't do anything to the original file, but alters slightly some punctuation in the printed copy. Certain symbols such as the @ and the & are transliterate commands and can be overcome by typing the symbol twice, such as & &. It mostly does only what you tell it to do. If you change your mind before it prints, FCTN 9 will take you back to the main menu. If it has already started printing, FCTN 4 will stop the print, and [ENTER] will return you to the main menu. I hope that this has alieviated some of your fears of FORMATTING. I hope you will give it a try.

"HIT ANY KEY TO CONTINUE"

JOYSTICK ADAPTER
by Rich Rehfeldt

Do you have some Atari Joysticks setting around doing nothing? Would you like to use them on the TI 99/4A ? Okay. here's a way to do it. This came from Compute Magazine, September 1983, and it works ok.

Here's a part list:
10 ea 914 Diodes RS 276-1122
1 ea Female Connector RS 276-1537
2 ea Male Connector RS 276-1537
1 Small Box RS 270-220



CALENDAR PROGRAM
Reviewed By Phil Storll
Typed By Cyndy E.

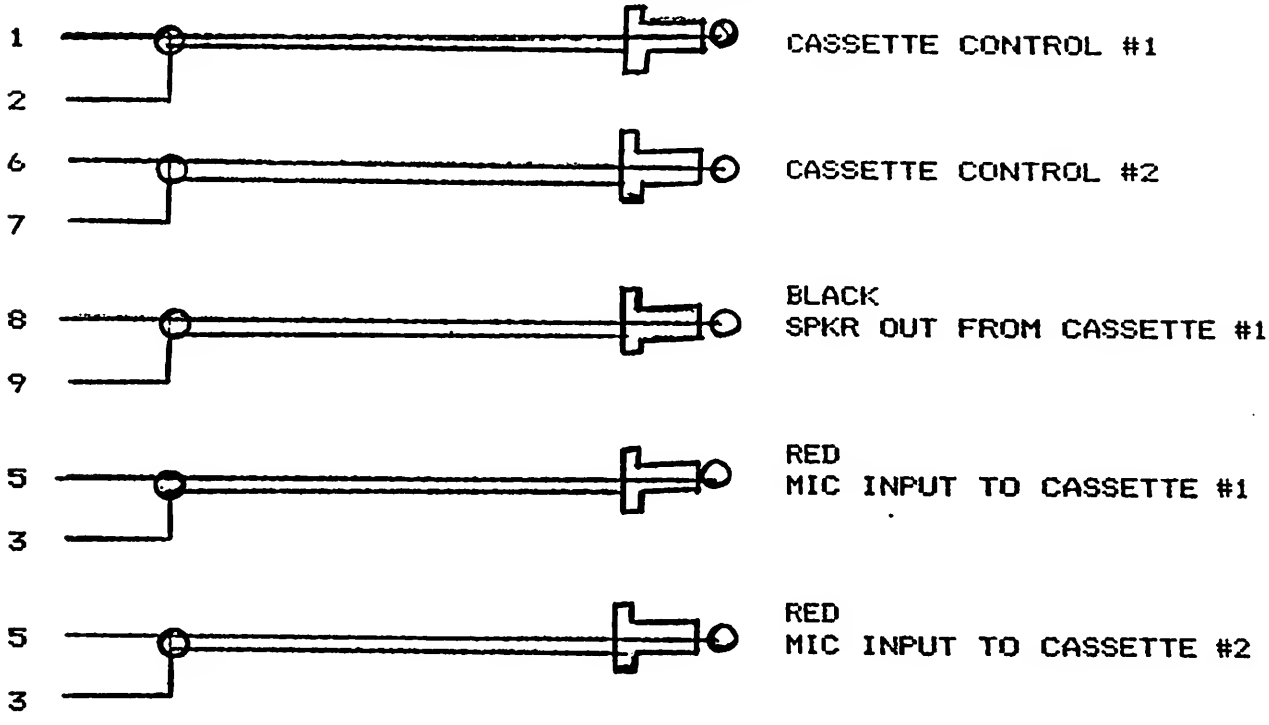
Calendar is one of those programs that no one should be without. With Calendar you can find out which day any Date of the month falls on the Gregorian calendar system in worldwide use today. It lets you create your own data file. Each data file can hold up to fifty Dates. It also shows you Holidays such as Labor Day, etc. Calendar requires Extended Basic.

This program is available through BeeJay Funware out of Denver and I will give you their phone number. I feel that I should not print their phone number without their permission.

25

BUILD YOUR OWN DUAL CASSETTE CABLE
by Rich Rehfeldt

Here is wireing for a TI Dual Cassette Cable, if you are in the building mood again. If you want a single cassette leave out # cable.



2 RED - RS 274-287 MINI PLUG

3 BLACK - RS 274-287 MINE PLUG

2 BLACK - RS 274-289 3/32 SUBMINI PLUG

1 9PIN FEMALE RS 276-1538 D TYPE

1 HOOD RS 276-1539

1 MINIATURE WITH SPIRAL WRAP SHIELD RS-278-752

---

* * * * * * COLOR BLEND * * * * * *

If you want pastel colors in your programs, make every other dot in your CHAR a one or a zero and then call the background color to be white (16). The program below will change the cyan color to a pastel shade

```
90 CALL SCREEN(16)
100 CALL COLOR(1,8,16)
110 CALL CHAR(32,"55AA55AA55AA55AA")
120 CALL CLEAR
130 GOTO 130
```

Try also 14, 12, 10, and 2 as the second number in line 100 for other colors. John Johnson, (Cedar Valey 99'ers Users Group)

TECTIP 1

by Harold Hoyt
and
Gene Breer

We recently connected an IBM Qumetrak DS/DD disk drive in place of the SS/DD Shugart that we had been using for more than a year without having any problems. We had used a 3 way connector to attach the PE box power supply to the external drive. It is supposed to be OK to do this if you are using two "low power" half height drives instead of the older original equipment drives that use a lot of current. When we accesed the drive to test it, the PE box power supply was unable to supply enough current to operate the drive and the file READ aborted, the program to got lost and the non volital RAM disk files became volitale and were lost, and had to be reloaded.

At this point, we felt that it was a good idea to design a test fixture to measure the power requirements of a disk drive. The design specifications we decided on that was easy to build, inexpensive and easy to use and understand. Unlike most hardware, it would be used only once in a while, when setting up a system. We purchased a drive expansion power cable and a drive power cable with wires attached. We could have just as easily started out with just the connectors, wire, inserts and a crimping tool, but it seemed easier this way.

Fig. 1 shows the result of our efforts. A male power connector is wired to a test fixture, which has 0.1 ohm resistors used as meter shunts in the +5 volt and +12 volt busses. The electric current requirement of the drives may be monitored by measuring the voltage drop across these resistors. Having such a low value of resistance allows current monitoring with negligible effect on the disk drive power voltages. Even at 1 ampere load current, the voltage drop is only 0.1 volt, yet connecting an inexpensive pocket voltmeter with a 250 millivolt range across the resistors allows reading up to 2.5 amperes since by ohms law, each millivolt drop across a 0.1 ohm resistor is caused by 10 milliamperes of current flowing through it. A low cost pocket voltmeter can often be obtained in the $20 range from places like Radio Shack. If a calibrated DC coupled oscilloscope with a 200 millivolt per centimeter deflection sensitivity is available, the transient performance of the power supply can be monitored. A 10 cm vertical deflection would be 2 amperes.

We added a range switch and range resistors so that a 1 milliampere meter movement with 43 ohms internal resistance will be properly calibrated to read 5 volts full scale on the +5 volt bus, 0.5 amperes full scale, 15 volts full scale on the +12 volt bus and 0.5 amperes full scale. When the Qume/IBM drive stepper motor was operating, the scale was pinned on the 12 volt current monitor. A DVM showed an average current of 0.75 amperes on the 12 volt supply.

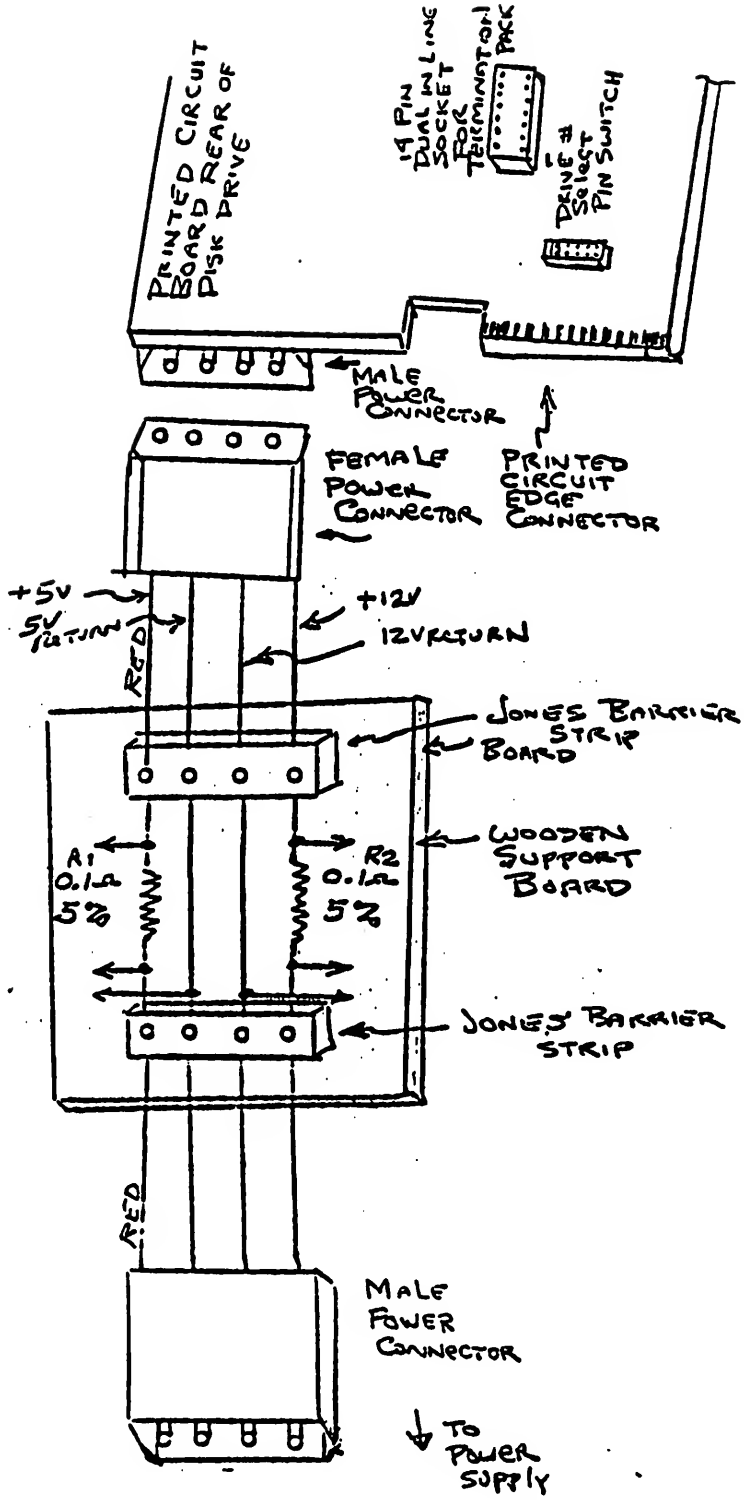


Fig 1 HSKR 2/9/88

DISK DRIVE TEST FIXTURE

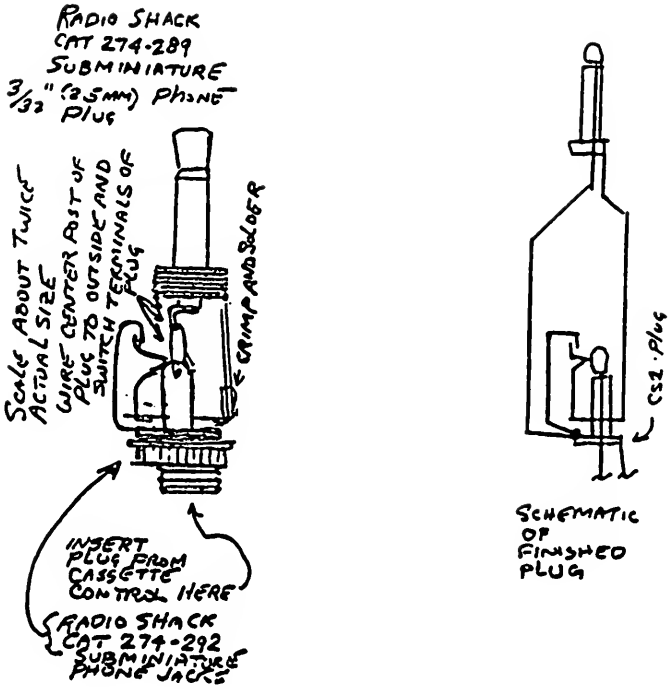
TECTIP 2

by Harold Hoyt
and
Gene Breer

Gene has been collecting TI/99-4A computers for his relatives and himself. The TI makes an excellent system for people that are just beginning. The TI is the only one I know of that has a user friendly cassette input. Gene recently purchased several recorders on sale at a discount house, only to find that the cassette cable wouldn't run the recorder. For most of us, this problem is just a memory, our cassette recorders and tapes are gathering dust.

The problem is that about half of the Cassette recorders in the world have an opposite polarity from the other half. The motor control circuit in the console requires the switch to have + voltage on the center electrode. To correct the problem, people have made an adapter that interchanges the inner and outer electrode on the end of the cassette motor control wire. The one in the accompnying drawing is a little fancier, in that it uses a jack with a switch so that the adapter can be left permanently in the cassette recorder. The computer seizes motor control only when the cassette control plug is inserted in the jack.

Most people ,after a while, don't even use the motor control. Some people don't even have the motor control input on their recorders. They leave the motor control wire hanging and manually conrol the recorder using the screen prompts to guide manual operation of the play and record functions.



ST. LOUIS 99ERS

THIS NEWSLETTER PUBLISHED MONTHLY, IS DEVOTED TO THE INTEREST OF THE ST. LOUIS 99ERS CLUB. MEMBERSHIP IS $15.00 PER YEAR. ARTICLES TO BE SUBMITTED SHOULD BE MAILED TO JAN KNAPP - 2318 RUCKERT, ST. LOUIS CO. MO. 63114 OR CALL 428-0752

MOUNT A "LED" ON SHUGGART MOD#SA200 DRIVE

by Willard Robinson

1. Mount LED in desired position on front panel of disk drive.

Note: Due to the various types and sizes of LED's available, it is impossible to fit a discription for mounting each. I recommend Radio Shacks LED cat# 276-018. A word of caution when drilling the hole in the front panel, be sure not to puncture and damage disk drive parts, and the area is clear enough for connecting wires.

2. Connect wires to LED. It is recommended to obtain a plug tha will slide onto the LED leads, but of you're good at soldering, you can solder a small gauge wire to each lead, being careful by heat sinking to keep from burning out the LED while soldering.

3. Connect other end of wire to J7 on the rear of the disk drive. J7 is located just to the left of pin 34 of the 34 pin controller connector J1. Again it is recommended to use a plug but if you want to solder it is O.K. The rear pin isthe negative lead (-).

4. Connect Disk Drive to the computer and operate. The LED should light while reading from or writing to th disk. If not try reversing the leads for proper polarity. No Dropping resistor is required, there is one built on the drive already.

28

****
* TECTIP 3 *
* by Harold Hoyt *
* and *
* Gene Breer *
****

We decided to take a real hard look at the PE Box power supply design when it refused to support 3 non-low current disk drives. We redrew the schematic to show the main components. We analyzed the power transformer (-2 version, multiple input voltage tap type which seems to be the production model.)

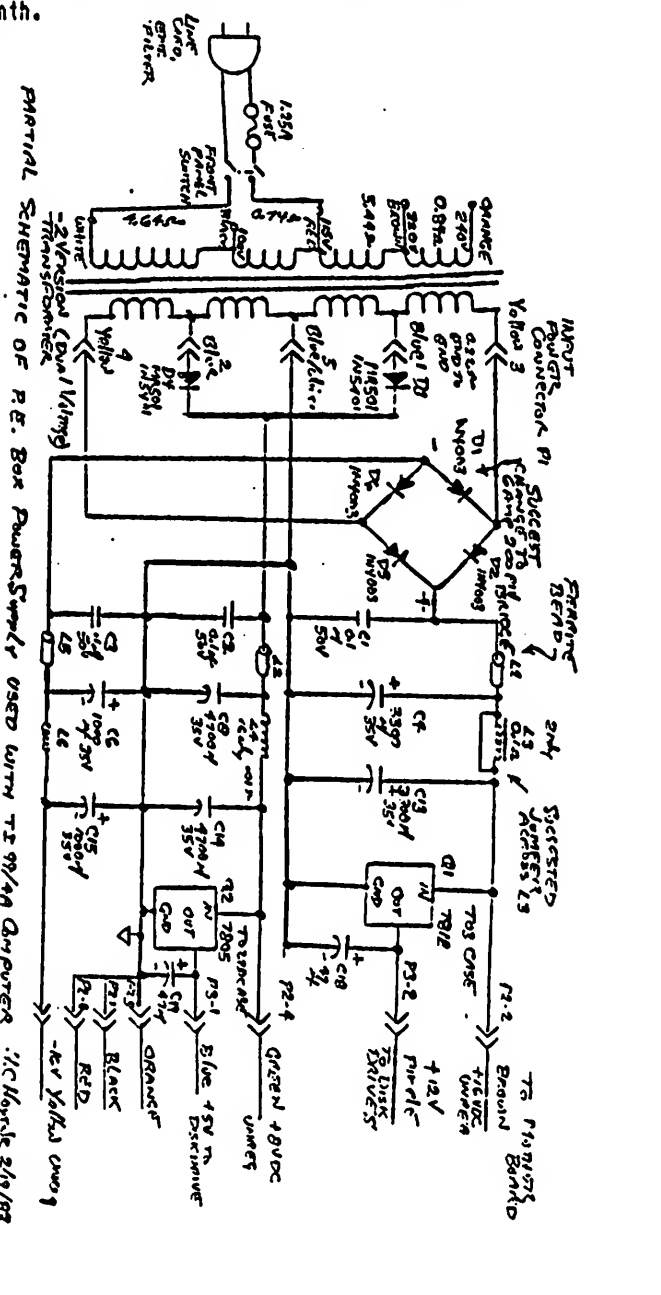
Our conclusions: The power supply should be redesigned. First, a power transformer of that size and type iron with proper utilization of it's windings should conservatively deliver 145 watts. Assuming a linear voltage regulator with loaded 16 volts DC, apportion 80 watts to the 12 volt DC side at 5 amperes output and the rest for the 5 volt output and the unregulated +16,+8,-16 buses for the PE Box cards.

We started on this project with the idea of eliminating external power supplies for outboard disk drives, or requiring the use of low power types in the half-height type. While we were working on this project, an article in March 1988 MICROpendium addressed a related problem. Dr. Eric W. Bray, M.D. Added an additional 12 volt regulator board and components to the existing supply. He tapped into the existing board to use the unregulated +16 volt output. His reasoon for doing this was to add a low power hard disk drive to the system. Although we are only part way done in our redesign, we feel that a properly structured redesign could address the hard disk problem too. Is there a cost and performance premium paid for the low power rating on the hard disk? We suspect so. On the basis of limited information, Hard disks motors requires 12 VDC at 1 to 5 amperes depending on the model. Some systems with hard disk power supplies designed in from the start boost the supply voltage on power up to get the disk up to operating speed faster. This would require special circuitry, which could be compatible with Dr. Gray's idea that the power supply for the hard disk should be separate, but run from the same transformer.

In any event, we found the existing power supply severly flawed. We wondered why they don't break more often! 1N4000 series 1 amp diodes should not be used in the 16 volt capacitor input filter. One is tempted to remove all power supply components, transformer and control board and start over. Our first attempt at breadboarding uncovered another problem. Simply putting heavy duty components in was not enough. The breadboard worked fine, but no better current rating than the TI design! We felt that we should stick to integrated circuit packaged regulators since they have built in fault/failure protection. Discrete component regulators, although easy to design, do not have "fail safe" built in. Any extra effort required to protect the expensive load components from melting is worth the effort. IC regulators have built in fold-back voltage-current safe area protection. Whoever designed the PE box power supply figured the transformer voltages wrong! the 16 volt unregulated voltage is actually over 24 volts! All of this makes unneccesary heat on the individual PE box boards. Fortunately, most of the boards don't draw much current anyways.

We are presently working on a scheme to reconnect primary and secondary windings on the PE box and adding a small transformer for the -16 volt unregulated supply. A lot of the capability of the transformer would be wasted by this scheme, but it seems a shame to replace such a big transformer. We would connect 120 volts to the 240 volt tap, running everything at half voltage, and then use 3/4 of the secondary winding to supply about 8 amperes max to the 16 volt "unregulated" bus and 12 volt regulated bus combination. The +8 volt unregulated will come from existing secondary taps.

If all this seems complicated, we apologize. Some of the above is a really needed note to other hardware people out there to alert them to the possibility of improving the PE box power. We hope to have another progress report next Month.



29

TECTIP 4

by Harold Hoyt and Gene Breer

This month we continue the analysis of the PE Box power supply design. We will have more to say next month about hardware change options. For this month, we will have some changes that will reduce heat generated in the existing power supply board and next month, a replacement design for this board will be described.

This advice is provided with the following caveat: **IF YOU ARE GOING TO ATTEMPT CHANGING THE WIRING IN YOUR COMPUTER, GET SOME COMPETENT HELP! WE WILL NOT FEEL RESPONSIBLE FOR ANYONE WHO HAS A MELTDOWN IN THEIR COMPUTER! WE WILL FEEL BAD, BUT NOT RESPONSIBLE!** Change #1 will partially correct an error in the transformer input voltage, which is much too high. Just making this change alone will reduce heat in the PE box to 120/140 of it's present value, a 14% reduction. If you have high current disk drives and/or other peripheral equipment that require more power, building a heavy duty replacement power supply board according to our instructions will help.

Get a set of PE box schematics. Remove all cards, including the "Fire Hose" connector card. Dissassemble the PE box by taking out all of the little black screws that hold it together. Don't lose these! Where can you find replacements? Carefully slide the front panel off, revealing the power supply in the left compartment, next to the fan. If you haven't done so by now, replace the fan with a quieter one. Radio Shack cat #273-242 will do fine.

Trace the power supply wiring, point by point using the schematic as a guide until you really know the circuit. What we are going to do is move the 120 volt AC input to 140 volt taps on the transformer. The 140 volt taps are not marked, but can be figured out by you by subtracting 240 volts (orange) from 100 volts (black). This is accomplished by changing the wires that connect the transformer to the on-off switch. The red (115 volt tap) is dis-connected and the orange (240 volt tap) is connected in it's place. also the white (0 volt tap) is disconnected and the the black (100 volt tap) is connected in it's place.

If you have the older PE box with the push button on-off switch follow these steps:

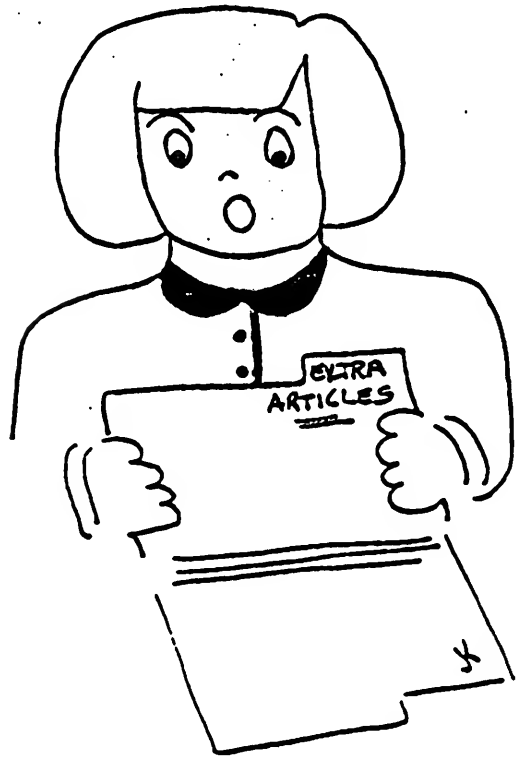
1. Remove red jumper quick disconnect terminal (115/120 volt tap on transformer) from front panel switch. 2. Connect orange (240 volt tap) wire from transformer to switch quick disconnect terminal where the red wire was connected previously. 3. Connect red wire to the jones-barrier strip terminal vacated by the orange wire. 4. Unsolder the white wire (0 volt tap on transformer) from the switch and cover with shrink sleeving to insulate it. 5. Solder an insulated jumper wire from the black wire (100 volt tap) on the jones-barrier strip to the switch terminal that previously held the white wire. If you have the newer PE box with the rocker type on-off switch follow these steps:

1. Remove the white wire (0 volt tap on transformer) from the switch and cover the quick disconnect with shrink sleeving to insulate it. 2. remove the black wire (100 volt tap) from the 1.25 ampere fuse block switch at the back of the console and connect it to the switch terminal vacated by the white wire. 3. Remove fuse and rotate it to the 240 volt position, which effectively disconnects the red(115/120 volt tap and connects the orange (240 volt tap) to the switch.

If you have properly made the above changes the 120 volt line will be connected to 140 volts of transformer winding. As a test of performance, the "-16 volt" unregulated output which measures a little over -24 volts before the change will measure about -21 volts. Still not what the designer intended, but it's the best we could do with the windings given us.

If the fan connection is left on the 115/120 volt tap, the actual fan voltage will be 98 volts, quite sufficient to run it, and the fan motor dissipation will also decrease. Fan motor speed and torque may drop slightly, but the fan moves more air than needed, even without the 14% heat reduction resulting from the changes.

EDITOR'S NOTE:
See diagram on opposite page.



HELP! MY FILE IS EMPTY —

TECHNOSPEAK

By Earl Raguse
(From the Feb.'88 ROM Newsletter)

Some people, especially government employees and computer programmers, speak in a language which is a bit strange to the mediocre mind, like mine. The following are some "tongue in cheek" examples. See if you can decipher them. They are often heard sayings or technospeakingly "Ancient adages impinging on ones otological appuratus with intermittent regularity".. Well, you get the idea! (If you ca't fugure them out, answers listed on page 10.)

1. Avian species of identical plumage congregate.

2. Freedom from encrustations of noxious substances is contigous to conformity with devine prescription.

3. Pulcritude possesses solely cutaneous profundity.

4. A superannuated canine is immune to indoctrination in innovative maneuvers.

5. Ululate not, over precipitated lacteal secretion.

6. All that coruscates with resplendence will not assay auriferous.

7. The existence of visible vapors from ignited carbonaceous materials confirms conflageration.

8. Mendicants are interdicted from elective recipiency.

9. Probity gratifies reflexively

10. Male cadavers are unyielding of fallacious testimony.

11. Inhabitants of vitreous edifaces ill-advisedly catapult petreous projectiles.

12. Ergonomia exclusive of diversion renders John a hebetudinous progeny.

13. He who cachinates ultimately, cachinates optimally.

14. Abstenation from speculatory undertaking precludes attainment.

15. Missles of lingeous and nonmetalic mineral consistency have potential for fracturing my osseous structure, but malicious appelations are eternally innocous.

REPRINT BAYOU BYTE 2/88

TECTIP 5

by Harold Hoyt
and
Gene Breer

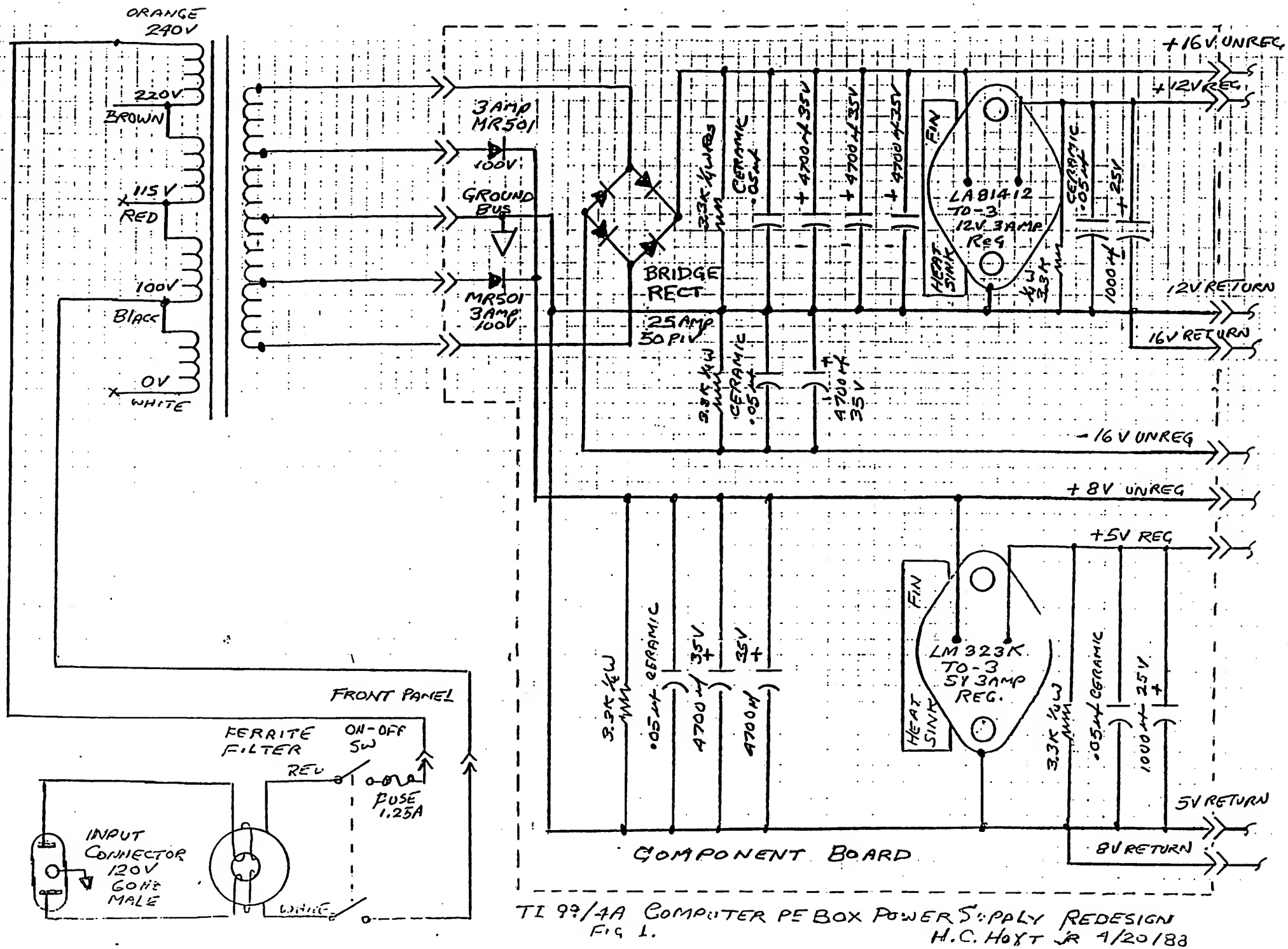
This month we finalize the PE Box power supply redesign. A replacement design for the PE box power supply board was shown last month and an extension of the idea suggested by Eric Bray, M.D. in the 3/88 MICROpendium is described.

This advice is provided with the following caveat: IF YOU ARE GOING TO ATTEMPT CHANGING THE WIRING IN YOUR COMPUTER, GET SOME COMPETENT HELP! WE WILL NOT FEEL RESPONSIBLE FOR ANYONE WHO HAS A MELTDOWN IN THEIR COMPUTER! WE WILL FEEL BAD, BUT NOT RESPONSIBLE! The transformer taps should be changed to the 140 volt taps as suggested in TECTIP4. Refer to Fig. 1.

The circuit is very similar to the TI version except that many parts have been eliminated and others changed to heavy duty types for higher current operation. A 25 ampere 50 PIV bridge replaces 2 fragile 1N4000 series rectifiers and 2 Motorola 3 amp MR501's. Capacitor Filtering is increased. Calculate filter capacitor requirements by I=C*Delta V/Delta t. With Delta V=1 Volt peak to peak ripple and 120 HZ Frequency I is 0.57 ampere. Assume 2 volts ripple and 1 ampere load current for each 4700 uf. Proper design minimizes heat in the regulator by having the worst case low voltage into the regulator at the ripple valley be 2 volts. The larger the ripple voltage, the more voltage required to prevent regulator dropout. Very large filter capacitors increase the heat in the transformer due to pulse peak current increase, but the transformer can handle heat a lot better than the linear semiconductor regulators. The bottom line: Lots of filter capacitors are desirable in well designed linear power supplies. Changing the transformer connection to supply the proper voltage for the regulator input voltage worst case design, low line voltage, max load current, ripple valley voltage 2 volts above desired output, results in much improved maximum load current before the built in overload protection shuts down the system. 7800 series and similar IC regulators protect themselves against destruction by adjusting the current limit downward as the voltage across the series pass element is increased. Since the transformer has too much secondary voltage the current limit is set much lower than needed.

Both the 12 volt and 5 volt regulators are changed to 3 ampere types. Catalog shopping for 3 amp types shows less than a 1$ difference between these and the wimpy 7800 series regulators. TO-3 cases and a Thermalloy 6002B-2 heat sink. (1.5" between mounting screw centers and 3/4" fins above the board can get rid of a lot of heat.)

Duplicate the 12 volt regulator sub-section if desired to squeeze an additional 2+ amperes for a hard disk drive. Add an additional 4700 uf capacitor to the second board for every 2 amperes of load expected. Put a 1000 uf capacitor on the output of the regulator to bypass output voltage transients. Put a ceramic 0.1 uf bypass capacitor on both the input and output of any IC regulator.



TI 99/4A COMPUTER PE BOX POWER SUPPLY REDESIGN
FIG 1.
H.C. HOYT JR 4/20/88

```
******************
*   TEC-TIP7     *
* ALPHA-LOCK FIX *
*      by        *
*  Harold Hoyt   *
*      and       *
*   Gene Breer   *
******************
```

Many people get tired of telling their kids they have to release the ALPHA-LOCK if both Joysticks are used. A fix has been around for some time where the ALPHA-LOCK may be in either position without affecting the joystick operation. A 1N914 or similar diode is placed in series with the ALPHA-LOCK key. A very good time to make this modification is when you are replacing a tired keyboard.

The change is a low risk operation compared to other computer modifications, although WE STILL RECOMMEND GETTING HELP FROM SOMEONE HANDY WITH ELECTRONICS IF YOU ARE INSECURE. Make the change to the replacement keyboard. Do all soldering to this keyboard before attaching it to the computer. Then you won't have to worry about an ungrounded soldering iron zapping your system. Many of us have purchased keyboards from Radio Shack, their part # 277-1023.

Stocks of this keyboard are spotty. Also, the keyboard was supplied by a bunch of different vendors and vary in quality from very good to very bad. If you are lucky, you can find original type keyboards with individual keys and gold plated switches. If you are unlucky, all you will find are poor quality pressure contact bubble switches that should wear out quickly and have a very spongy feel unsatisfactory to typists. Check out the keyboard action before you buy.

The ALPHA-LOCK modification suggested requires adding a single diode to the keyboard. Since there is a variety of printed circuit layouts for the keyboard, it is hard to be specific about a physical description of where to place the diode. Refer to the key matrix schematic that comes with the Radio Shack keyboard, which is reproduced here with the diode penciled in. Check the connections to the keyboard with an ohmmeter to be certain that you know where the connector pins are. Some of these naughty people placed the marking stripe on the pin 15 end rather than the pin 1 end. Cut the trace going to pin 6 conveniently close to the connector. (Tech notes purchased at the Chicago Fair suggest putting the diode in the connector cable, but that isn't rugged enough for me.) Solder the 1N914 diode cathode (ring or striped end) to the connector and the anode end to the cut trace. Install the modified keyboard, following CAVEMAN'S directions in the January newsletter. Check to see that you get upper case characters with the ALPHA-LOCK depressed and lower case with the ALPHA-LOCK in the "up" position. Run a test program (listing below) or a game to check that both joysticks totally ignore the ALPHA-LOCK key position.

```
100 CALL JOYST(1,X1,Y1)
110 CALL JOYST(2,X2,Y2)
120 PRINT X1;Y1;X2;Y2
130 GOTO 100
```

33

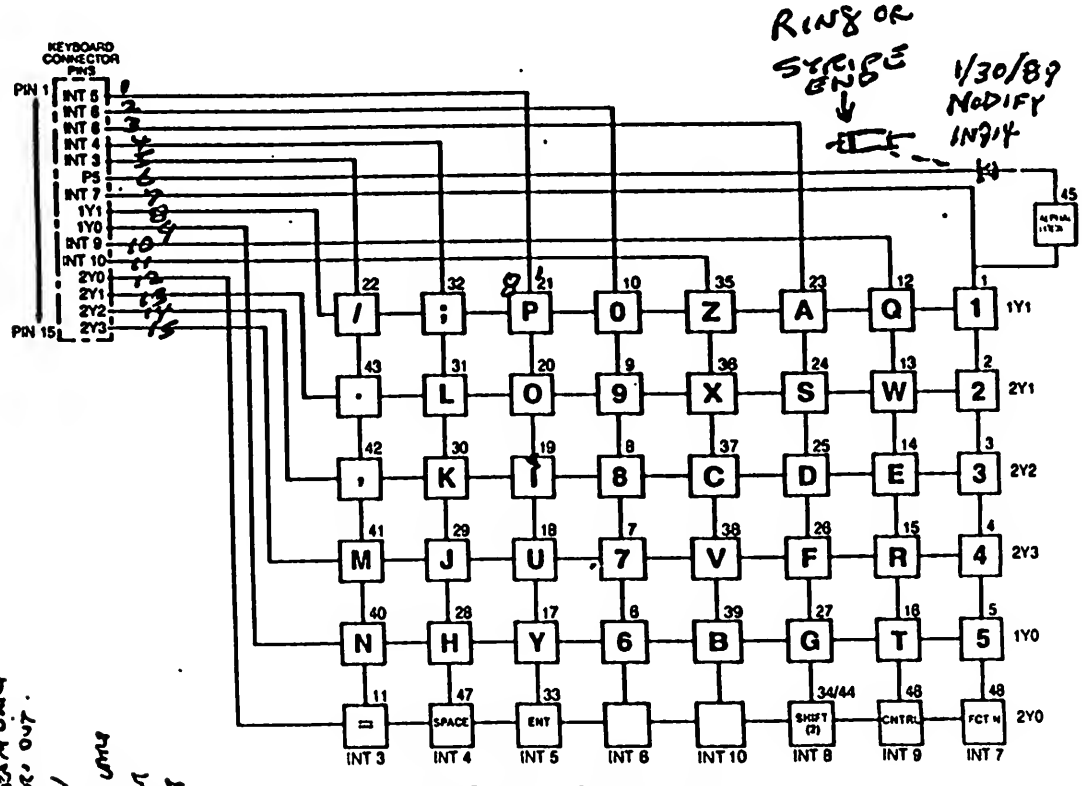
ARCHER®

Cat. No. 277-1023

ALPHA-NUMERIC KEYBOARD

Originally for TI 99/4 Computer

A high-quality QWERTY keyboard with standard typewriter stagger format, electrically arranged in X-Y matrix. All keys SPST momentary contact except for ALPHA LOCK, which is an alternate action switch. Electrical connections may be made to connector (AMP 1-640441-5) or directly to PC board.



KEYBOARD SCHEMATIC

Custom Packaged in USA for Radio Shack
A Division of Tandy Corporation, Fort Worth, TX 76102